НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»
УДК 004.056

«До захисту допущено»
В.о. завідувача кафедри
_____ М.В.Грайворонський
“___”_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Виявлення APT-атак на рівні операційних систем за допомогою методів машинного навчання

Виконав : студент 2 курсу, групи ФБ-81мп
(шифр групи)

Гребенюк Михайло Сергійович
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доц. Барановський О. М.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (освітньо-професійна програма) – 125 Кібербезпека («Системи, технології та математичні методи кібербезпеки)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)

«___»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Гребенюк Михайло Сергійович
(прізвище, ім'я, по батькові)

1. Тема дисертації: Виявлення APT-атак на рівні операційних систем за допомогою методів машинного навчання

науковий керівник дисертації: Барановський Олексій Миколайович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___» _____ 2019 р. № _____

2. Термін подання студентом дисертації 10.12.2019 р.

3. Об'єкт дослідження: поведінка користувачів, процеси в операційних системах, файли, віртуальні пристрої, інтернет з'єднання та зв'язки між ними

4. Вихідні дані: покращення виявлення APT-атак на рівні операційних систем з застосуванням методів машиного навчання

5. Перелік завдань, які потрібно розробити:

а) огляд існуючих літературних джерел щодо моделей APT-атак, методів виявлення APT-атак в операційних системах

b) аналіз та вибір методів машиного навчання, побудова моделі даних діяльності користувачів в оперпційних системах

c) розробка експерементального програмного комплексу, визначення ефективного аглоритму машиного навчання для виявлення APT-атак

6. Орієнтовний перелік ілюстративного матеріалу: 20 – 25 ілюстрацій

7. Орієнтовний перелік публікацій: стаття в журналі "THEORETICAL AND APPLIED CYBERSECURITY", на тему "Виявлення APT-атак в операційних системах".

8. Дата видачі завдання 04.09.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Опрацювання літературних джерел	27.09.2019	Виконано
2	Отримання технічного завдання	02.10.2019	Виконано
3	Аналіз методів машиного навчання	11.10.2019	Виконано
4	Розробка програмного комплексу	14.11.2019	Виконано
5	Написання магістерської дисертації	28.11.2019	Виконано
6	Підготовка до захисту	03.12.2019	Виконано

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Робота обсягом 99 сторінки містить більше ніж 23 ілюстрацій, 16 таблиці, та 22 літературних посилань.

Метою даної кваліфікаційної роботи є визначення найбільш ефективних методів з використанням машинного навчання для виявлення APT-атак та застосування цих метоів для покращення IDS.

Об'єктом дослідження є поведінка користувачів, процеси в операційних системах, файли, віртуальні пристрої, інтернет з'єднання та зв'язки між ними.

Предметом дослідження є методи детектування APT-атак на рівні операційних систем.

Методами дослідження було обрано: опрацювання літератури за даною темою, використування машиного навчання а саме алгоритми кластеризації графів, експериментальне дослідження з побудовою тестового середовища та програмних засобів.

Результати роботи можуть застосовуватись при процесі виявлення APT-атак на рівні операційних систем, та зборі детальної інформації що до системної діяльності, тобто імплементуватись у IDS, також при розслідуванні інцидентів кібербезпеки.

Кібер загрози, машине навчання, операційні системи, процеси, APT, алгоритми кластеризації, Linux.

ABSTRACT

The work includes 99 pages, 23 figures, 16 tables and 22 literary references.

The aim of this qualification paper is to identify the most effective methods using machine learning to detect APT attacks and to apply these methods to improve IDS.

The objects of the study are user behavior, processes in operating systems, virtual device, files, Internet connections and relations between them.

The subject of the research is methods of detecting APT attacks at the operating system level.

Research methods were chosen: analysis of the literature, methods of machine learning, namely graph clustering algorithms, experimental research using software tools.

The results can be applied in the development of the detecting APT attacks at the level of operating systems, gathering detailed information regarding system activities, implemented to IDS. Also it can be used in the investigation of cyber security incidents.

Cyber threats, machine learning, operating systems, processes, APT, clustering algorithms, Linux.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

APT-атака — англ. advanced persistent threat або розвинена стійка загроза. Термін широко використовується як кібер-загроза (кібер-атака) для комплексної, цілеспрямованої і ефективної атаки на критичні IT-інфраструктури. Вперше використовувалося командою реагування на комп'ютерні інциденти в Lockheed Martin (LM-CIRT).

APT-група – група хакерів, що здійснювала APT-атаки за для власної вигоди.

Машинне навчання — англ. Machine learning or ML. Це клас методів штучного інтелекту, характерною рисою яких є не пряме рішення задачі, а навчання в процесі застосування рішень безлічі подібних завдань.

Горізонтальний мережевий рух — англ. Network Lateral Movement, відноситься до прийомів кібернападів, які використовуються для поступового переміщення через мережу, використовується коли шукаються ключові дані та активи або кінціві ціль для атаки.

Виявлення аномалій поведінки мережі (NBAD) — англ. Network behavior anomaly detection, забезпечує один з підходів до виявлення загрози для мережевої безпеки. Може бути частиною IDS.

Система виявлення вторгнень — англ. intrusion detection system (IDS).

Прецедент — пара «об'єкт, відповідь».

Сукупність прецедентів — це пари «об'єкт, відповідь», або навчальна вибірка.

ВСТУП

З розвитком інформаційних та комунікаційних технологій збільшується кількість способів провести вдалі атаки на віддалені інформаційно-технологічні інфроструктури.

Питання виявлення злонамірних дій стає все більш актуальним в наш час. Проте із зростанням різноманіття сервесів, протоколів, форматів данних, об’ємів данних стає складніше виявити атаку до початку несанціонованих дій.

Стандартним рішенням стало впровадження алгоритмів машинного навчання у IDS (системи виявлення вторгнення). Цей підхід вдало використовується для виявлення вже відомих атак типу: Probe, R2L, DoS і т. д.

Впровадження алгоритмів машинного навчання на великі інформаційні-інфраструктури з великою щільністю нестандартизованого трафіка сповільнює та ускладнює аналіз. Саме це сприяє вдалому впровадженню APT-атак: складно (але іноді можливо) відрізнити сервісну або користуваціку діяльність від нестандартної злонамірної діяльності в інформаційному середовищі.

Актуальність роботи зумовлюється тим, що різноманніття алгоритмів машинного навчання та методів аналізу данних дає можливість поліпшити методи виявлення APT-атак. На сьогоднішній день APT-атаки є одними з найнебезпечніших загроз для великих IT-інфраструктур.

Метою роботи є визначення ефективних методів з використанням машинного навчання для виявлення APT-атак, та покращення застосування в IDS.

Для досягнення даної мети необхідно виконати такі **завдання**:

1) Проаналізувати існуючі дослідження в області застосування APT-атак
2) Проаналізувати існуючі дослідження в області методів виявлення APT-атак.

3) Запропонувати покращення обраного метода виявлення APT-атак з застосуванням машинного навчання.

Об’єктами дослідження є системна діяльність та інформаційні об’єкти, процес обміну даними у інформаційному середовищі, механіки реалізації APT-атак.

Предметом дослідження є методи виявлення APT-атак в операційних системах, алгоритми машинного навчання та їхні якісні показники.

Методами дослідження було обрано: опрацювання літератури за даною темою, класифікації APT-атак за низкою критеріїв, аналіз статистичних показників існуючих методів виявлення атак даного типу, експериментальне дослідження з використанням існуючих бібліотек для збору обробки та аналізу інформації, семпли даних які містять в собі інформацію про користувацьку та злонамірну діяльність .

Наукова новизна одержаних результатів: ефективні методи виявлення APT-атак на рівні операційних систем з використанням методів машиного навчання.

Практичне значення полягає у можливості використання результатів роботи для покращення ефективності IDS, та отримані більш детальної інформації що до системної діяльності.

1 РІШЕННЯ ТА МЕТОДИ ВИЯВЛЕННЯ APT-АТАК

В цьому розділі розглядаються сучасні моделі APT-атак, методи виявлення, їх переваги та недоліки. Визначається інформаційне середовище в якому виявлення атак буде найбільш ефективним.

1.1 Моделі APT-атак

Одну з перших поширених моделей APT-атак в 2011 запропонували працівники компанії Lockheed Martin [2]. Вони ввели новий термін Cyber Kill-Chain, який був частиною їх моделі Intelligence Driven Defense [1] для ідентифікації та запобігання процесам кібер-вторгнення.

Ця модель визначає, що повинні зробити зловмисники для досягнення своїх цілей, атакуючи мережу, витягуючи дані і підтримуючи присутність в мережі. Завдяки цій моделі відомо, що блокування злодіїв на будь-якому етапі розриває весь ланцюжок атаки.

За наведенною нижче моделю кінцева точка є неминучою точкою, через яку йдуть всі атаки, отже, зупинка атаки на цьому етапі істотно підвищує шанси на протидію різноманітним кібер загрозам.

Очевидно, що імовірність успіху буде вище, якщо зловмисники будуть зупинені на ранніх етапах.

Крім того, кожне вторгнення, залишає сліди впродовж всього ланцюжка, - це шанс краще пізнати дії зловмисників та використовувати цю інформацію у подальшій обороні. Чим краще досліджено дії зловмисників, їх методики, тим більше ймовірність побудувати більш ефективну оборону на деякий час.

Стандартна модель Cyber-Kill Chain наводить основні етапи (рисунок 1.1):

Рисунок 1.1 – Модель Cyber-Kill Chain

- Зовнішня розвідка (анг. External Reconnaissance):

Цей етап може бути визначений як фаза вибору мети, виявлення особливостей організації, специфічних вимог в даній галузі, вибір технологій, вивчення активності компанії в соцмережах. Зловмисники намагається зрозуміти, які методи атаки будуть працювати з найбільшим ступенем успіху або які з них буде лечге всього здійснити з точки зору інвестицій ресурсів та часу.

- Озброєння (анг. Weaponization):

Включає в себе різні форми: експлуатація веб-додатків, стандартні або спеціально виготовлені шкідливі програми (анг. exploit), уразливості в парсерах різних документах (PDF, XML, DOC, XLs або інших форматів документів).

Визначаються атаки типу watering hole (у цій атаці визначається набір веб-сайтів, які часто відвідують члени цільової групи, заражається один або декілька з них шкідливим кодом, врешті хтось з групи відвідає скомпроментований сайт). Зазвичай вони готуються з дуже конкретними знаннями про цільову аудиторію.

- Доставка (анг. Delivery):

Передача необхідного (шкідливого) контенту або за ініціативою жертви (користувач скачює з шкідливого сайту програми, документи і т.д..). Або з ініціативи зловмисників заргузки шкідливого інформаційного

навантаження через POST, PUT, GET запити. Сюди також можна выднести SQL, XML injection або code injection.

- Експлуатація (анг. Exploitation):

Після доставки на комп'ютер або будь-який інший пристрій користувача, необхідний (шкідливий) контент розгортається, встановлюється в оточенні.

Як правило, це відбувається при використанні відомої вразливості, для якої раніше був доступний патч. Дуже рідко це може бути zero-day вразливості.

Статистика показує, що в більшості випадків (в залежності від мети) зловмисникам не потрібно нести додаткові витрати на пошук і експлуатацію невідомих вразливостей.

- Інсталяція (анг. Installation):

Найчастіше інсталяція відбувається на тлі якихось зовнішніх з'єднань. Зазвичай шкідлива програма ховається в цих операціях, непомітно проникаючи на кінцеві точки мережі, до яких можна отримати доступ. Потім зловмисники можуть контролювати цей шкідливий додаток без відома жертви залишаючись непомітним.

- Отримання управління (анг. Command and Control (C2)):

На цьому етапі зловмисники починають контролювати пристрої жертви. Контроль може відбуватися через наступні канали: DNS запити на стороні DNS сервери [8], Internet Control Message Protocol (ICMP), запити на різноманітні веб-сайти, соціальні мереж, месенджери тощо.

В результаті, зловмисники передають команди: що робити далі та яку інформацію збирати. Використовуються для збору даних наступні методики: знімки екрану, контроль натискання клавіш, перехват паролів, моніторинг мережі на облікові дані, збір критичного контенту і документів. Часто призначається проміжний хост, куди копіюються всі дані, а потім вони стискаються / шифруються для подальшої відправки.

- Виконання дій на стороні жертви (анг. Actions on Objectives):

На фінальному етапі зловмисники відправляють зібрані дані на проміжний сервер і / або виводить з ладу (шифрує) IT-активи під час свого перебування в мережі жертви. Після цього проводяться заходи щодо виявлення інших цілей, розширення своєї присутності всередині організації та (що найважливіше) вилучення даних.

Згодом ланцюжок дій повторюється. Особливістю Cyber-Kill Chain моделі є те, що вона кругова, а не лінійна. Як тільки зловмисники проникли в мережу, вони знову починають цей ланцюжок всередині мережі, здійснюючи додаткову розвідку.

Треба мати на увазі, що хоча методологія однакова, але при знаходженні всередині мережі злодії використовують інші методи для етапів внутрішньої ланцюжка, ніж в разі, коли вони знаходяться поза мережею. Це обумовлено тим, що після проникнення злодіїв в мережу, вони стають інсайдерами (користувачами з певними правами в мережі), а це заважає фахівцям компанії з безпеки підозрювати атаку.

Виділяють такі поняття як зовнішня (попередньо розглянута) та внутрішня Cyber-Kill Chain (в разі горізонтального мережевого руху зловмисників в мережі).

Поєднання зовнішньої і внутрішньої Cyber-Kill Chain називається розширеною моделлю Cyber-Kill Chain (рисунок 1.2). Вона включає в себе додаткові пункти:

- Внутрішня розвідка (анг. Internal Reconnaissance):

На цьому етапі злодії мають доступ до пристроя одного з користувачів та передають на нього дані, общуковують локальні файли, мережні папки, історії браузерів і тд.

Мета - це з'ясувати, як цей пристрій може допомогти просканувати мережу і як перейти до більш цінних ресурсів.

- Внутрішня експлуатація (анг. Internal Exploitation):

Цей етап характерезується використанням вразливостей не пропачених програмних сервисів, web додатків або зміні облікових даних чи використання за замовчуванням. Це все дозволяє піти зловмисникам від підконтрольного пристрою користувача до серверів з використання підвищення привілеїв, горізонтального мережевого руху та контролю остаточних цільових пристроїв.

Етапи розвідки та озброєння в зовнішньої Cyber-Kill Chain моделі можуть розтягуватись на великий час. Також наведені етапи е важкими в перехоплені: зловмисники проводять ці етапи без з'єднання з мережею.

Рисунок 1.2 – Розширена модель Cyber-Kill Chain

Більш формальну модель запропонувала компанія Mandiant [3] під назвою APT Lifecycle Model (рисунок 1.3). Вона є більш формальною та прикладною, фахівці будували її на сучасних тенденціях та звітах. Зазначається, що в більшості випадків нападники використовують прості методи вторгнення в мережі жертв.

Рисунок 1.3 – Mandiant APT Lifecycle Model

APT Lifecycle Model складається з наступних етапів:

- Розвідка (анг. Reconnaissance):

Mandiant визначає, що на цьому кроці зловмисники ідентифікують осіб через яких можна потрапити в мережу. Цільов аудиторія: починаючи від головного керівництва до рядового помічника адміністратора.

- Початкове втручання в мережу (анг. Initial Intrusion into the Network):

Слід відзначити повністю збіжність з етапом “Доставка” (анг. Delivery) моделі Cyber-Kill Chain. Цей етап визначається великою ймовірністю використання соціальної інженерії та спірального цільового фішенгу (анг. spear phishing).

- Встановлення бекдора в мережі (анг. Establish a Backdoor into the Network):

Зловмисники намагаються отримати адміністративний домен з обліковими даними (як правило, у зашифрованому вигляді) з цільової компанії та передачу файлів за мережу. Mandiant ідентифікував випадки, коли зловмисники дістали та розшифрували облікові дані протягом декількох хвилин. Після цього зловмисники підвищують прівелегії в мережі та встановлюють декілька бекдорів в мережі з різними конфігураціями.

Mandiant спостерігає такі характерні риси в більшості шкідливих програм:

- Використання шифрування та обфускацію даних для передачі мережею, що ускладнює аналіз трафіку C2 (анг. Command and Control).
- Використання вбудованих бібліотек, якщо вони доступні, для зменшеня розміру програм.
- Використання типових облікових данних користувачів, щоб злонамірна діяльність співпадала з діяльністю користувача.

- Отримання облікових даних користувача (анг. Obtain User Credentials):

Зловмисники отримують доступ до більшості системи за допомогою дійсних повноважень. Вони часто націлюються на домен контролерів для отримання облікових записів користувачів і відповідних хеші паролів. Зловмисники також отримують місцеві дані від скомпроментованих систем.

Слід зазначити, що цей етап частково перетинається з етапом “Отримання управління” (анг. Command and Control (C2)) моделі Cyber-Kill Chain.

- Встановлення різноманітних програмних засобів (анг. Install Various Utilities):

Зловмисники використовують утиліти для виконання

загалиних системних завданнь адміністрування. Вони мають програмне забезпечення яке має використовуватись для встановлення бекдорів, скидання паролів в дефолтні, отримання електронної пошти з серверів, отримання списку запущених процесів, аналізу логів і багато інші завданнь. Однак ці утиліти часто зустрічаються в системах які не містять бекдорів. Тому Mandiant робе припущення, що зловмисники встановлюють свої утиліти але використовують дійсні облікові дані.

- Ескалація привілегій, горізонтальний мережевий рух, зкачування даних (анг. Privilege Escalation, Lateral Movement, Data Exfiltration):

Після встановлення надійного каналу зв'ялку та контролю, злодії починають збирати такі дані як e-mail та додатки до них, файли з робочіх станцій або проєктні файли з серверів.

У більшості випадків, потрібна інформація стискається з використанням архівних утіл RAR, tar, zip та шифрується паролем. Потім інформацію надсилають на проміжний тимчасовий сервер (анг. staging servers) підконтрольний зловмисникам. Цей етап співпадає з останім етапом стандартної моделі Cyber-Kill Chain.

Зловмисники можуть виводити дані безліччю способів з мережі, але Mandiant засвідчує, що в більшості випадків використовуються загальна тактика на ряді жертв. Найбільш поширеними методами є:

- Використання тимчасового сервера;
- Шифрування та стискання даних;
- Видалення скомпроментованих данних з тимчасових серверів після виведення їх з мережі.

Тимчасові сервери можна визначити по наявності та слідів роботи архівних утіліт, після криміналістичного огляду.

- Поркащення контролю (анг. Maintain Persistence):

Зловмисники будуть додавати зусилля до присутності у мережі, додаючі додаткові бекдори, підбираючи більш витончені інструменти та розповсюдження контролю.

Цей етап відсутній в стандартній моделі Cyber-Kill Chain але детально розкритий в розширеній моделі Cyber-Kill Chain.

Враховуючи швидкій ріст технологій, зміни в корпоративних інфраструктурах, вдосконалення методів нападу важко визначити эдине стабільне рішення для виявлення APT-атак. До цього питання треба підходити виключно зогляду корпоративної інфраструктури.

Окремої уваги заслуговує рішення відомої компанії MITRE, вона побудувала фактичну модель під назвою ATT&CK (див. [4] та [5]), яка описує які дії може здійснити зловмисник в мережі в залажності від наявності різноманітних операційних систем, програмних засобів, серверів і т.д.. Наведена модель є більш детальною та наближеною до дійсності ніж попередньо наведені Cyber-Kill Chain та APT Lifecycle Model.

ATT&CK модель складається з ланцюжків тактик, які може вибрати зловмисник виходячи з інформаційного середовища компанії. Кожна тактика належить відповідній категорії, наразі MITRE нараховує їх дванадцять:

- Initial Access
- Execution
- Persistence
- Privilege Escalation
- Defense Evasion
- Credential Access
- Discovery
- Lateral Movement
- Collection
- Command and Control
- Exfiltration

- Impact

Категорії тактик об’єднуються в ATT&CK Matrix[5] (рисунок 1.4).

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Commonly Used Port	Automated Exfiltration	Data Destruction
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	BITS Jobs	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Communication Through Removable Media	Data Compressed	Data Encrypted for Impact
External Remote Services	Command-Line Interface	Account Manipulation	AppCert DLLs	Binary Padding	Brute Force	Browser Bookmark Discovery	Distributed Component Object Model	Clipboard Data	Connection Proxy	Data Encrypted	Defacement
Hardware Additions	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	Domain Trust Discovery	Exploitation of Remote Services	Data Staged	Custom Command and Control Protocol	Data Transfer Size Limits	Disk Content Wipe
Replication Through Removable Media	Control Panel Items	AppInit DLLs	Application Shimming	CMSTP	Credentials in Files	File and Directory Discovery	Logon Scripts	Data from Information Repositories	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Structure Wipe
Spearphishing Attachment	Dynamic Data Exchange	Application Shimming	Bypass User Account Control	Clear Command History	Credentials in Registry	Network Service Scanning	Pass the Hash	Data from Local System	Data Encoding	Exfiltration Over Command and Control Channel	Endpoint Denial of Service
Spearphishing Link	Execution through API	Authentication Package	DLL Search Order Hijacking	Code Signing	Exploitation for Credential Access	Network Share Discovery	Pass the Ticket	Data from Network Shared Drive	Data Obfuscation	Exfiltration Over Other Network Medium	Firmware Corruption
Spearphishing via Service	Execution through Module Load	BITS Jobs	Dylib Hijacking	Compile After Delivery	Forced Authentication	Network Sniffing	Remote Desktop Protocol	Data from Removable Media	Domain Fronting	Exfiltration Over Physical Medium	Inhibit System Recovery
Supply Chain Compromise	Exploitation for Client Execution	Bootkit	Exploitation for Privilege Escalation	Compiled HTML File	Hooking	Password Policy Discovery	Remote File Copy	Email Collection	Domain Generation Algorithms	Scheduled Transfer	Network Denial of Service
Trusted Relationship	Graphical User Interface	Browser Extensions	Extra Window Memory Injection	Component Firmware	Input Capture	Peripheral Device Discovery	Remote Services	Input Capture	Fallback Channels		Resource Hijacking
Valid Accounts	InstallUtil	Change Default File Association	File System Permissions Weakness	Component Object Model Hijacking	Input Prompt	Permission Groups Discovery	Replication Through Removable Media	Man in the Browser	Multi-Stage Channels		Runtime Data Manipulation
	LSASS Driver	Component Firmware	Hooking	Control Panel Items	Kerberoasting	Process Discovery	SSH Hijacking	Screen Capture	Multi-hop Proxy		Service Stop
	Launchctl	Component Object Model Hijacking	Image File Execution Options Injection	DCShadow	Keychain	Query Registry	Shared Webroot	Video Capture	Multiband Communication		Stored Data Manipulation
	Local Job Scheduling	Create Account	Launch Daemon	DLL Search Order Hijacking	LLMNR/NBT-NS Poisoning and Relay	Remote System Discovery	Taint Shared Content		Multilayer Encryption		Transmitted Data Manipulation
	Mshta	DLL Search Order Hijacking	New Service	DLL Side-Loading	Network Sniffing	Security Software Discovery	Third-party Software		Port Knocking		
	PowerShell	Dylib Hijacking	Path Interception	Deobfuscate/Decode Files or Information	Password Filter DLL	System Information Discovery	Windows Admin Shares		Remote Access Tools		

Рисунок 1.4 – MITRE ATT&CK Matrix

Кожна тактика має детальний опис який складається з наступних блоків (тегів):

- Короткої інформації;
- Переліку APT-груп, які використовували конкретний підхід;
- Платформи на якій тактику можна впровадити (Ubuntu, macOS, Windows);
- Системні права (User, Administrator, SYSTEM);
- Джерело даних (Process monitoring, Process command-line parameters);
- Версія (1.0);
- Зменшення впливу (анг. Mitigation). Наприклад блокувати деякі команди в консолі;
- Виявлення. Наприклад: дії інтерфейсу командного рядка можуть бути проаналізовані шляхом належного реєстрації виконання процесу з аргументам.
- Ефект. Що відбудеться після реалізації данної загрози.
- Перелік літературних джерел.

Наведені тактики вказують, який шлях обере зловмисник. Тактики описують дії таким чином, щоб вони не залежали від конкретних шкідливих програм та інструментів зловмисника. Перевага цього підходу полягає в тому, що вона описує поведінку, за допомогою інструментів віддаленого доступу, скриптів або взаємодії в інтерфейсі командного рядка на пристрояї компанії, не прив'язуючсь до певних “протилежних” шкідливих програм та інструментів, які з часом можуть змінюватися.

Також компанія MITRE побудувала CAR (Cyber Analytics Repository) [8], це база даних аналітики, розроблена на основі моделі MITER ATT & CK. Аналітика, що зберігається в CAR, містить наступну інформацію:

- гіпотеза, яка пояснює ідею після аналітики;
- інформаційний домен або первинний домен, в якому аналітична система призначена для роботи (наприклад, хост, мережа, процес);
- посилання на методики та тактики ATT&CK;
- опис псевдокоду про те, як може бути реалізована аналітика;
- тест, який можна запустити, щоб провести аналітику;

В загальному випадку, аналітична інформація в CAR, може виявити поведінку зловмисника в рамках моделі ATT&CK.

Опираючись на наведену модель різноманітні компанії будують свої системи захисту, логувань, відслідковування подій та ідентифікаціїї вторгненя або використовують вже існуючі засоби які підходять до їх інфроструктур. В багатьох випадків це можуть бути open source проекти які підтримуються community. Наприклад BZAR (Bro/Zeek ATT&CK-based Analytics and Reporting) [6].

1.2 Методи виявлення APT-атак на рівні операційних систем

Детектування атак можно поділитина дві групи:

- детектування мережевої діяльності;
- детектування діяльності на рівні операційної системи.

З огляду ATP технік будь яка мережева направлена злонамірна активність буде мати тотожно відповідне відображення на атакованих операційних системах. Розглядаючи виявлення злонамірної діяльності на рівні операційної системи слід зазначити – що цей напрям охоплює більшу структурну частину в попередньо наведених моделях: Cyber-Kill Chain, APT Lifecycle Model, ATT&CK Matrix.

Наприклад навіть під час Network Lateral Movement зловмисникам потрібно буде відкривати мережеве з’єднання на контрольованій машині, та запускати відповідні програми, мати доступ взаємодіяти з мережевими сокетами.

Виходячи з точки зору системного проектування та програмування визначаються наступні види діяльності в операційних системах (на прикладі linux-подібних систем):

- операції з файлами:
    - створення;
    - модифікування;
    - видалення;
- операції з утілітами:
    - системних;
    - зовнішніх;
    - консольних/десктопних
- використання бібліотек:
    - системних;
    - зовнішніх;
- операції з архівами;

- керування правами доступа:
    - керування власними правами доступу;
    - керування груповими правами доступу;
    - підвищення привілеїв;
- керування користувачами:
    - створення/видалення;
    - модифікування налаштувань;
- операції з приладами:
    - створення/видалення;
    - читання/запус у символьнині або блочні прилади;
    - монтування файлових систем;
- робота з сокетами:
    - UNIX, DATAGRAM, TCP, UDP, IPv4, IPv6, etc.;
    - читання/запис, відкриття/закриття;
- використання кріптобібліотек;
- взаємодія з системними демонами (syslogd, networkd, etc.)
    - start;
    - stop;
    - reload;
    - restart;
    - force-reload;
- операції з procfs, sysfs (модифікування/зчитування налаштувань/стану ядра, операційної системи, доступ до даних з перефірійних приладів);
- створення процессів, форків та керування ними;
- керування пам'яттю;
- керування файловими дескрипторами;
- запити в системне ядро (iostl, netlink, etc.)

Велика кількість джерел інформації потребує швиткодійні методи обробки/фільтрацції інформації та надає можливість ефективного ідентифікування загроз.

1.3 Аналіз існуючих методів виявлення APT-атак

Оглядаючи існуючі дослідження та інструменти виявлення загроз виділяються наступні методи детектування APT-атак на рівні операційних систем:

- Log Analysis – обробка логфайлів з серверів, програмних засобів, баз даних etc., використвоуючи детектування спеціальних поміткок, часу події, самої події в логах. В новітніх рішеннях використовується машинне навчання;
    - Приклади:
        - Splunk;
        - Sumologic;
        - Papertrail;
        - Elastic;
    - Недоліки: в більшості випадків це централізовані системи – на обробку данних треба час (в деяких рішеннях до 1 дня), детектування загроз обмежується множиною повідомлень, яку генерують виділені програмні засоби. Наприклад, якщо зловмисник почне змінювати стандартні python бібліотеки, утіліти, створювати дамп файлової системи (десь у /tmp/.darkdir), відкривати нові мережеві з’єднання – це не буде відображатись у логаг з бази данних або веб сервера. Тобто система детектування не спрацює.

- System info monitoring – збір усієї(або часткової) системної інформації описаної в розділі 1.2 та вивід у зручному для користувача виді.
    - Приклади:
        - FileMon;
        - Process Monitor;
        - RegMon;
        - DataSecurity Plus;
        - Process Hacker;
    - Недоліки: спрацьовує людський фактор. Знаючи архітектуру, встановлене програмне забезпечення можно видавати злонамірну діяльність за системну або користувацьку. Людина фізично не може обробити велику кількість інформації, вона може не звернути увагу: на назнайомий процесс, миттєву операцію в декілька секунд, схожі на користувацьких процесс данні (але кординально різні по функціоналу), змінені права доспупів до файлів, зміну каталогів, виклик бібліотек etc. Збереження подібних данних потрибує великих об'ємів пам'яті (за деклілька секунд можливо генерування понад 2-х гігабайтів), це факт максимально ускладнює процесс аналізу.
- Traffic Analysis або NBAD як частина IDS.
    - Приклад Snort.
    - Недоліки: може не спацьовувати, якщо керування атакою йдеться через замасковані канали даних, наприклад Telegram.
- Security information and event management (SEAM).

  Обробка логів та евентів, пошук атак за визначиними правилами та кореляцією подій
    - Як і в Log Analysis множина подій та логів обмежується кодом який це підтримує, отже зломисник може це обійти.
- Використання пасток (traps):

- Приклад: Palo Alto Networks
- Недоліки: спрацьовує тільки на відомі експлойти, детектування 0-day неможливе.

- File and process scanning: базується на пошуку відомих або схожих сигнатур.
  - Приклад: розробка від компанії FireEye – File Content Security
  - Недоліки: неможливе розпізнування модифікованих або обфускованих об'єктів які мають – незнаємі сигнатури.

Висновки до розділу 1

Проаналізовано характерні риси, техніки здійснення APT-атак на онові діяльності світових APT-груп.

Розглянуто наступні моделі опису атак:

- Cyber-Kill Chain;
- Розширена модель Cyber-Kill Chain;
- Mandiant APT Lifecycle Model;
- ATT&CK Matrix;

Після детального аналізу визначенно, що розширена модель Cyber Kill Chain повністтю описує життевий цикл атак та підкреслює їхню ітеративність та циклічність на відміну від моделі Mandiant APT Lifecycle Model.

На сьогоднішній день найінформативнішим описом APT-атак є модель ATT&CK Matrix, слід відзначити що цей проект Mitre має сильну підтримку та використовується в багатьох дослідженнях.

Розглядаючи наведені моделі виявленно, що переважна структурна частина атак здійснюється на серверах або PC користувачів. З огляду на це випливає висновок, що детектувати APT буде більш вигідно провести на рівні

операційної системи: чим більше злонамірної діяльності, тобто процесів, тим більше інформації для виявлення вторгнення.

В рамках APT порівнюючи інформативність данних можна з впевніностю відзначити, що інформативність діяльності в операційних системах буде більше або дорівнювати інформативності мережевої діяльності. Це чітко проглядається на модялях Cyber-Kill Chain та Mandiant APT Lifecycle Model.

Виявлення APT-атак на рівні операційної системи — ефективно.

Для Виявлення використовуються наступні методи:

- Log Analysis;
- System info monitoring;
- Traffic Analysis;
- Security information and event management;
- Traps;
- File and process scanning.

Наведені методи з невеликою ймовірністтю або частково детектують описанні в попередньо-згаданих моделях атаки через приведені недоліки:

- Обмеженне детектування системної діяльності;
- Складність в обробці інформації та визначенні котристувацької, системної та злонамірної діяльності;
- Великий час на обробку інформації;
- Пошук інформації за шаблонами.

Маємо проблему: не ефективне детектування на рівні операційної системи. Необхідно дослідити покращення або доповнення існуючих методів.

Для подальшого дослідження виділено методи виявлення APT-атак на рівні операційної системи.

2 ОПТИМАЛЬНИЙ МЕТОД ВИЯВЛЕННЯ APT-АТАК НА РІВНІ ОПЕРАЦІЙНИХ СИСТЕМ

В данному розділі розглядаються проблеми обробки системної діяльності. Визначаються ефективні моделі представлення діяльності користувачів в операційних системах. Обераються оптимальні методи обробки інформації.

2.1 Формалізація проблеми виявлення діяльності системних користувачів

Виявлення APT-атаки на рівні операційної системи складна задача, вона потрибує розуміння будови ОС та загальний процес роботи утіліт, сервисів, драйверів, користувацьких програм і т. д..

Визначемо загальну потребу: відрізняти та вияляти діяльність операційної системи та користувача від злонамірної діяльності.

Визначемо загальні проблеми виявлення на рівні операційної системи:

- Різноманітність операційних систем, немає уніфікованного методу збору даних, оскільки системи фундоментально відрізняються в будові.
- Різноманітність представленя данних;
- Обмежена інформація про діяльність в системі:
    - Більшість інструментів дозволяє збирати тільки обмежену інформацію: (pid, username, params, etc).
    - Втрачання інформації через великі проміжки часу в збиранні інформації: (наприклад проміжок буде 2с. За цей час процесс виконає свою роботу та припине існування).
    - Відкидання ключової інформації через великий обсяг данних

- Складність в обробці інформації та визначенні котристувацької, системної та злонамірної(аномальної) діяльності;
    - Відсутність детермінованості: операційні сестеми весь час потребують оновлень. Мінімум це патчінг вразливостей в програмному забезпеченні. В більшості це регулярні оновлення, встановлення нового програмного забезпечення, заміна старих програм новими. Тобто інформація яку збирали пів року тому буде координально відрізнятися від нової.
    - Наявність неінформативних даних. Наприклад діяльність Chrome генерує велику кількість запитів до системних бібліотек, через те що кожна вкладинка є окремим процессом в системі див. рис. 2.1.
    - Ймовірність зплутати активність користувача з системною активністю. Наприклад діяльність системного адміністратора з діяльністю скриптів з /etc/init.d/*. Адміністратор може використовувати нові утіли, встановлювати нові сервіси.
    - Велика швидкість накопичення данних: (приблизно 1гб за 10 хвилин на PC).
- Різноманітність параметрів та залежностей в данних.

Виходячи з описаних проблем, видно, що для обробки даних не підійдуть шаблоні рішення з попередньо визначиними правилами. Отже треба використовувати динамічні методи, які будуть ефективними при будьякій зміні вхідних даних.

gdbus	3417	3421	j3	10u	REG	259,1	32768	9463032	/home/j3/.local/share/gvfs
chromium-	3449		j3	cwd	DIR	259,1	4096	9437186	/home/j3
chromium-	3449		j3	rtd	DIR	259,1	4096	2	/
chromium-	3449		j3	txt	REG	259,1	156686192	2493398	/usr/lib/chromium-browser/
chromium-	3449		j3	DEL	REG	0,26		465	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	25913104	2232306	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	DEL	REG	0,26		385	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	18748872	3279465	/usr/share/fonts/opentype/
chromium-	3449		j3	DEL	REG	0,26		112	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	622592	9962621	/home/j3/.config/chromium/
chromium-	3449		j3	mem-w	REG	259,1	262144	9962600	/home/j3/.config/chromium/
chromium-	3449		j3	mem	REG	259,1	756072	3279527	/usr/share/fonts/truetype/
chromium-	3449		j3	DEL	REG	0,26		396	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	DEL	REG	0,26		451	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	1809656	2232315	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	240272	2236527	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	225992	2237363	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	1636360	2232310	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	2496856	2232307	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	28672	9962772	/home/j3/.config/chromium/
chromium-	3449		j3	mem-r	REG	259,1	77824	9962763	/home/j3/.config/chromium/
chromium-	3449		j3	DEL	REG	0,26		54	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	DEL	REG	0,26		52	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	353824	3279661	/usr/share/fonts/truetype/
chromium-	3449		j3	DEL	REG	0,26		39	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	41080	2365175	/var/cache/fontconfig/9456
chromium-	3449		j3	mem	REG	259,1	84432	2365151	/var/cache/fontconfig/04aa
chromium-	3449		j3	mem	REG	259,1	238328	2359341	/var/cache/fontconfig/365b
chromium-	3449		j3	DEL	REG	0,26		28	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	524656	10098179	/home/j3/.config/chromium/
chromium-	3449		j3	mem	REG	259,1	290904	2492169	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	662008	2234774	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	DEL	REG	0,26		100	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem-r	REG	259,1	16384	9962608	/home/j3/.config/chromium/
chromium-	3449		j3	DEL	REG	0,26		66	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	4202496	10098183	/home/j3/.config/chromium/
chromium-	3449		j3	mem	REG	259,1	1056768	10098182	/home/j3/.config/chromium/
chromium-	3449		j3	mem	REG	259,1	270336	10098181	/home/j3/.config/chromium/
chromium-	3449		j3	mem	REG	259,1	45056	10098180	/home/j3/.config/chromium/
chromium-	3449		j3	DEL	REG	0,26		55	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	7944	2364603	/var/cache/fontconfig/99e8
chromium-	3449		j3	mem	REG	259,1	32880	2364599	/var/cache/fontconfig/a6d8
chromium-	3449		j3	mem	REG	259,1	20584	2365157	/var/cache/fontconfig/2cd1
chromium-	3449		j3	mem	REG	259,1	3416	2365154	/var/cache/fontconfig/0d8c
chromium-	3449		j3	mem	REG	259,1	3144	2365156	/var/cache/fontconfig/1ac9
chromium-	3449		j3	mem	REG	259,1	12392	2365161	/var/cache/fontconfig/385c
chromium-	3449		j3	mem	REG	259,1	8640	2365171	/var/cache/fontconfig/767a
chromium-	3449		j3	mem	REG	259,1	17256	2365174	/var/cache/fontconfig/8801
chromium-	3449		j3	DEL	REG	0,26		59	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	mem	REG	259,1	426456	2233160	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	31784	2622818	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	mem	REG	259,1	114184	2236551	/usr/lib/x86_64-linux-gnu/
chromium-	3449		j3	DEL	REG	0,26		61	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	DEL	REG	0,26		46	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	DEL	REG	0,26		45	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	DEL	REG	0,26		44	/dev/shm/.org.chromium.Chr
chromium-	3449		j3	DEL	REG	0,26		43	/dev/shm/.org.chromium.Chr

Рисунок 2.1 – Діяльність программи Chromium Web Browser

Доцільно використовувати методи машинного навчання. Вони дозволяють динамічно оброблювати великі обсяги інформації з великою ентропією, виділяти поведінкові моделі, класифікувати або кластеризувати датасети, в нашому випадку розрізняти діяльність користувача та операційної системи від злонамірної діяльності. Що до вибора алгоритма машиного навчання див. розділ 2.3..

2.2 Побудова моделі структури даних для обробки системної діяльності

Перш ніж обирати алгоритм машиного навчання, шукати аномалії в поведіці користувачів, детектувати злонамірну активність, треба структурувати дані та визначити природу цих даних. Визначивши природу даних задача підбору алгоритма значно спроститься.

Для дослідження були взяті найбільш популярні Linux-подібні системи. Слід зазначити, що структура зв'язків процессів в Windows системах майже подібна до Linux систем.

Пропонуэться 2 загальних моделі представлення процесів:

- Звичайний підхід: відображення множини процессів в множену n-мірного простору, де для кожно процесу буде існувати відповідна точка.
- Підхід з урахуванням властивостей процессів: відображення множени процессів в граф, де для кожного процесу буде існувати відповідна вершина.

Загальна модель представлення даних у вигляді точок у n-мірному просторі: кожну метрику можно представити одною характеристикою процессу. Наведемо наступні метрики:

- Час роботи процесса в секундах або мілісекундах;
- Середня кількість процессів-нащадків на визначеному проміжку часу;
- Кількість запитів до файлів:
  - Читання;
  - Запис;
  - Читання та запис;
- Кількість запитів до приладів;
- Кількість запитів до сокетів;

- Середня кількість використаної пам’яті на визначеному проміжку часу;
- Середня кількість слотів файлових дескрипторів на визначеному проміжку часу;
- Кількість потоків;
- Числове представлення назви процеса;
- Ідентифікатор користувача;
- Витрачені процесорні ресурси.

Додатково розширити цей список можна в залежності від операційної системи. Для більш детальної інформації що до linux-подібних систем можна звернутись до джерела [10].

Загальна модель представлення даних у графовому вигляді: структура процесса в Windows та linux-подібних системах наближенна до графового дерева, якщо розглядати їх разом без залежності від користувачів. Кожен процесс має батьківський процесс та процеси нащадки. Також кожен процесс може мати доступ до файлів з різним виду доступу (читання, запис або читання разом з записом), доступ до системних приладів (/dev/null, /dev/ptmx, /dev/USB, /dev/tty), доступ до сокетів (UNIX, TPC, UDP, DOMAIN). Ці данні легко уявити як листки графового дерева. Представлення реального процеса в linux-подібних системах наведенно на рис. 2.2, а саме процесс mongod сервіса, pid – 4179.

З рисунку видно що mongod має доступ до файлів в яких зберігаються дані (/data/db/collection*), таблиць індексів (/data/db/index*), лок-файлів (/data/db/lock*). Також відзначається доступ до системних бібліотек (/usr/lib/x86*).

Отже можна дійти висновку, що чим більше програмний засоб несе функціонального навантаження, тим більше активності буде відображено у графовому представлені.

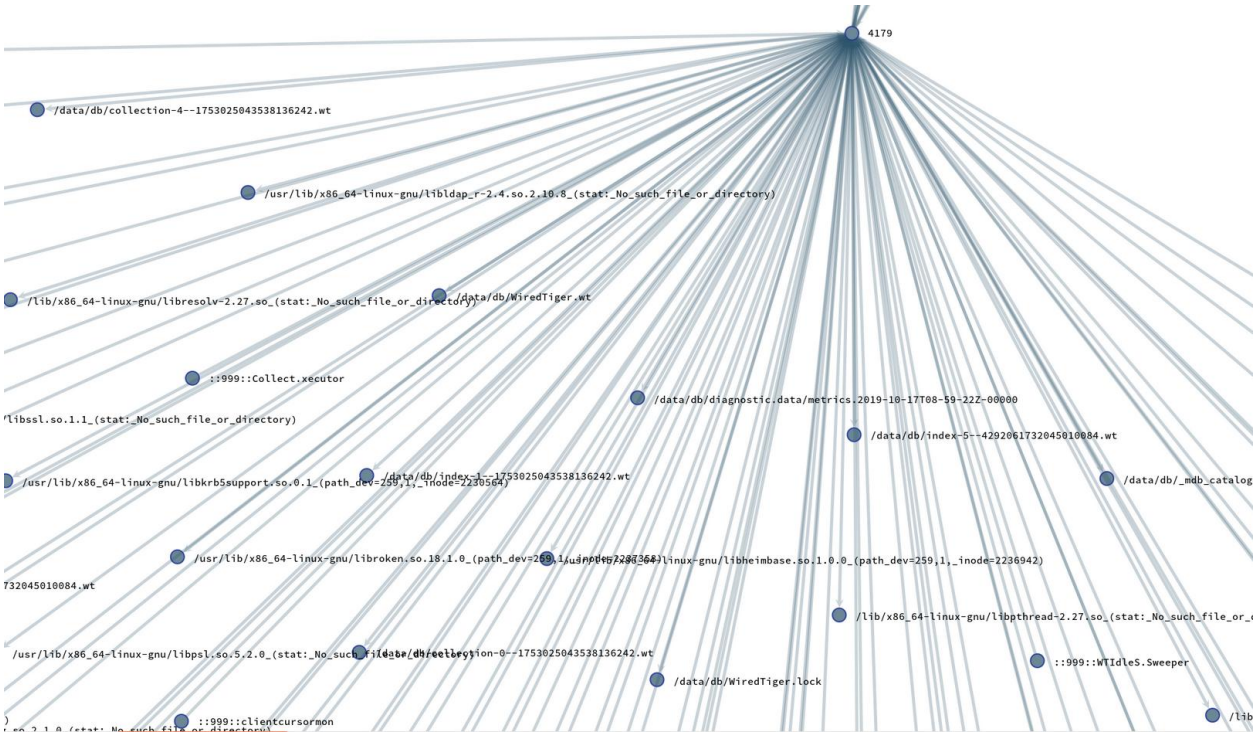

Рисунок 2.2 – Діяльність mongod сервіса в linux середовищі

При розгляданні сукупності процессів в системі древовидне представлення втрачається, оскільки декілька процессів можуть мати доступ до одного й того ж файла або пристрою.

Схематично структура даних буде мати вигляд наведений на рис 2.3. Детальне реальне представлення наведене на рис. 2.4 та рис. 2.5. Для будови графа збирались дані протягом 30 хвилин з операційної ситеми Ubuntu16.

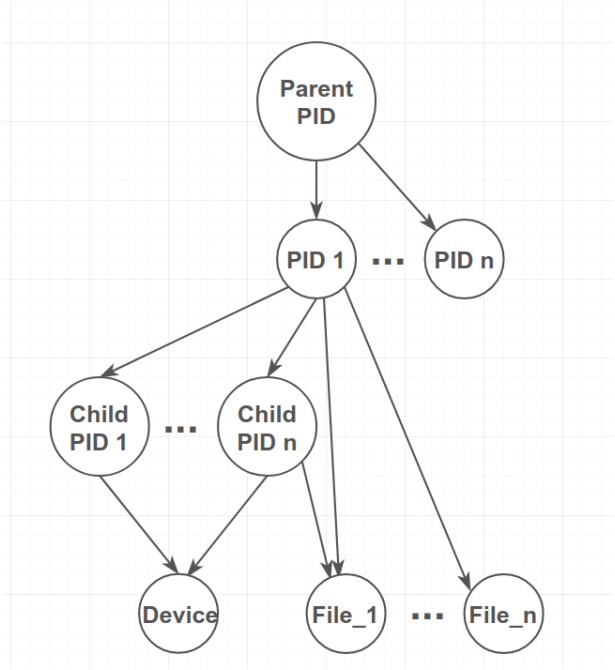

Рисунок 2.3 – Схематична графова структура даних

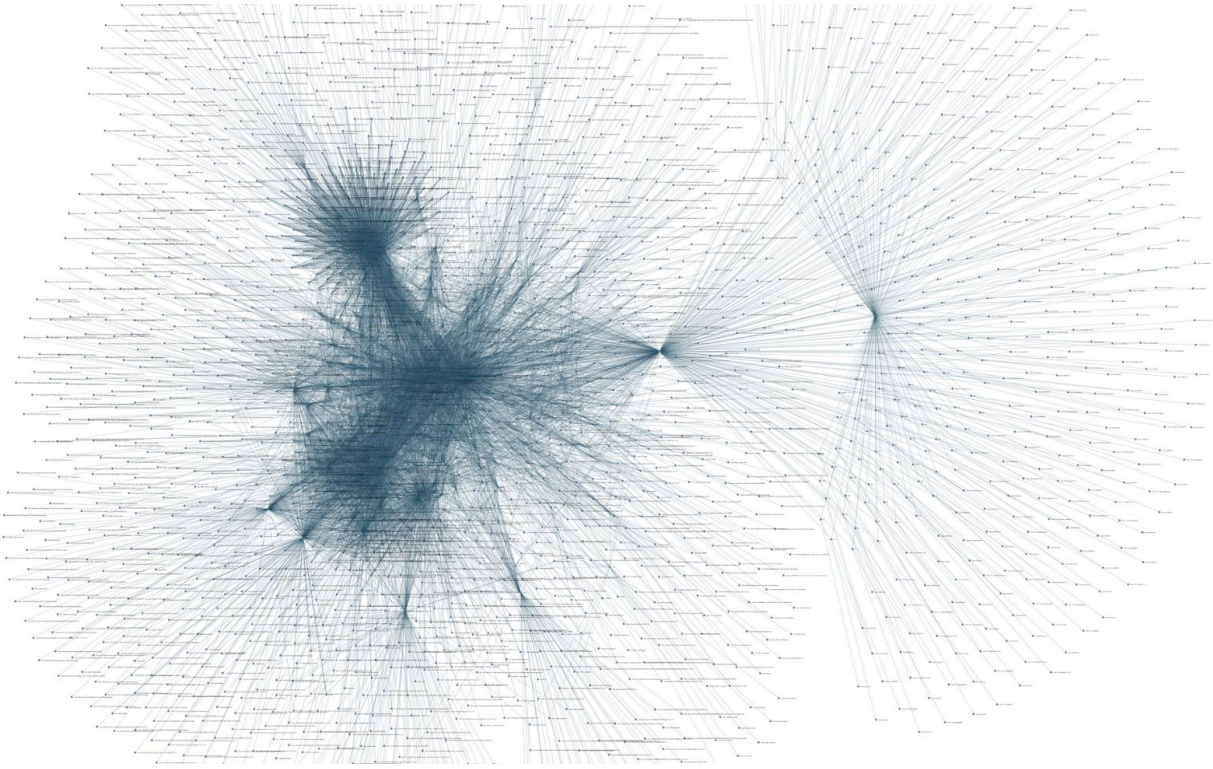

Рисунок 2.4 – Відображення системної діяльності в Ubuntu16

Збереження вз’язків процес-процес, процес-файл реалізується за допомогою направлення ребра. Вага ребер розраховувалась з часу протягом

якого існував зв’язок між вершинами. На рисунках вага відображенна прозорістю.

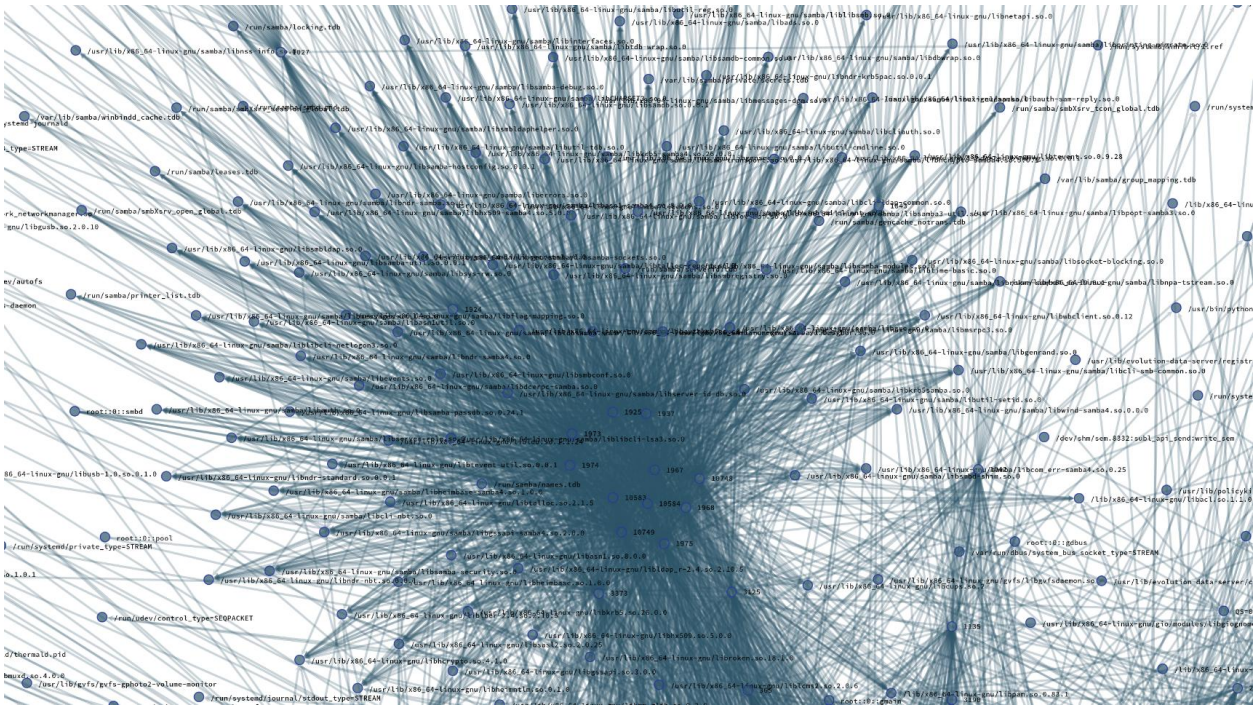

Рисунок 2.5 – Наближене відображення діяльності сукупності процессів в Ubuntu16

Використовуючи наведену графову модель представлення процессів відтвореми злонамірну діяльність, а також додамо в модель активність користувача, характерну для системного адміністратора або программіста. Результат відображенно на рис. 2.6 та 2.7, злонаміна діяльність відображається помаранчевим кольором та включає в себе мережеву активність та запуск наступних програм: ping, namp, ssh, tracepath, sh, su, sudo, vi, less, mongo.

Для створення зображення була використана бібліотека graph-tool[9] написанна на python3 та C++.

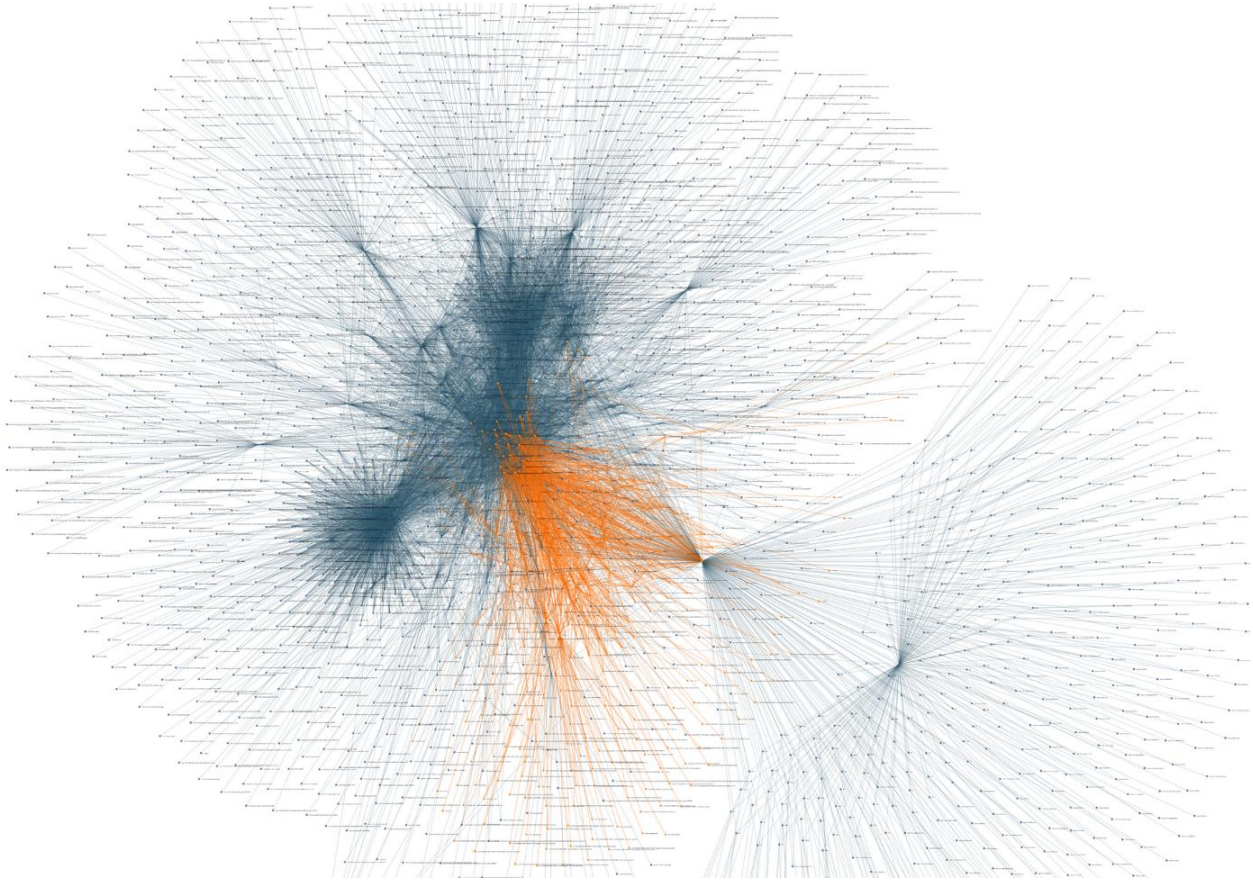

Рисунок 2.6 – Графове відображення злонамірної діяльності

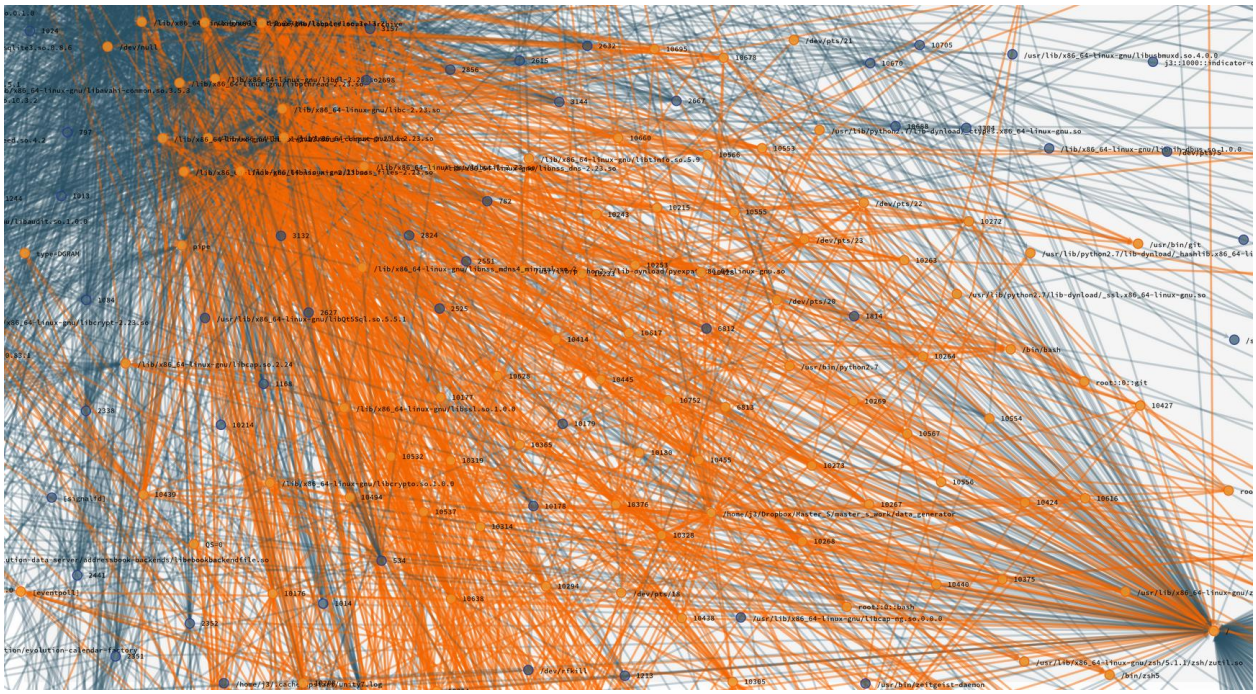

Рисунок 2.7 – Наближене графове відображення злонамірної діяльності

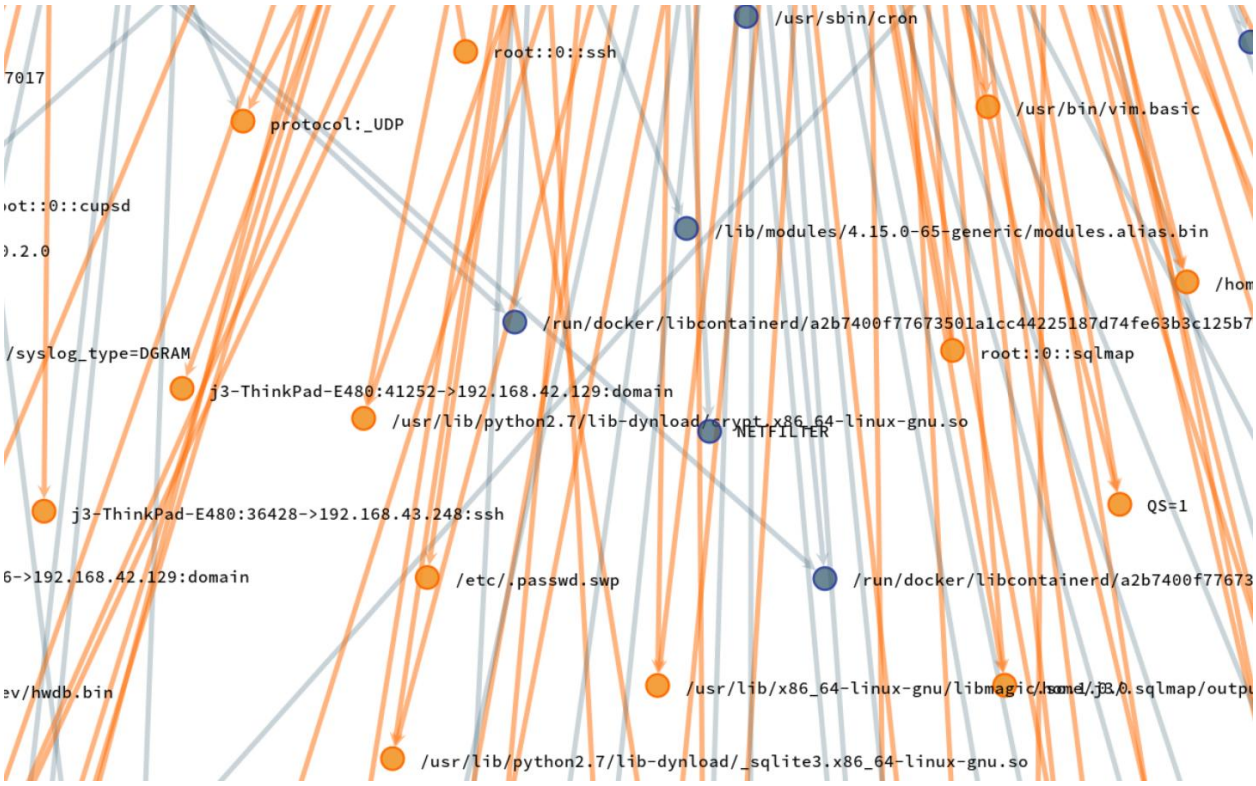

Рисунок 2.8 – Наближене графове відображення злонамірної діяльності

З рисунків 2.6, 2.7 та 2.8 чітко прослідковується об’єднання злонамірної діяльності в один цільний підграф. Це відбувається коли процеси були запущені з однієї оболонки (наприклад sh) або мають один і той самий батьківський процес (в n-му коліні батьків), наприклад експлуатація вразливості в web-сервері дозволила відкрити reverse shell. Результати свідчить про те, що в такому представлені дані легко будуть піддаватись групуванню за обраними рисами.

2.3 Вибір ефективних методів для обробки системної діяльності

Базуючись на висновках та переліку проблем наведених у розділі 2.1, а також моделях даних наведених в розділі 2.2 доцільно говорити про використання методів машиного навчання. Враховуючи велику різноманітність алгоритмів використаєм джерела [11] та [12] для побудови класифікації.

Класифікація по типам задач :

- Задача класифікації – отримання категоріального відповіді на основі набору ознак. Має кінцеву кількість відповідей (як правило, це «так» або «ні»): чи є на фотографії червоний собака, чи є на зображенні обличя панди , чи спроможний кліент виплатити кредит.
- Задача кластеризації – розподіл даних на групи за визначеними характеристиками: поділ усіх клієнтів фіінансового банку за рівнем платоспроможності, віднесення супутників до тієї чи іншої категорії, розподіл системних процесів на системні або користувацькі.
- Задача регресії – це результат або прогнозування на основі вибірки об'єктів з різними ознаками чи параметрами. На виході ці алгоритми видають результат – дійсне число.

  Наприклад: ціна нового автомобіля, вартість електроніки, затрачені ресурси програмою, очикуваний дохід компанії від нового проекту або кількість заражених пристроїв новим вірусом.
- Задача виявлення аномалій – відділення сукупності аномалій від групи стандартних випадків. На перший погляд вона зпівпадає з класифікацією але є одна істотна відмінність: аномалії – явище рідкісне. Навчальних даних, з допомогою яких можно виявити такі об'єкти нема або недостатньо. Тому методи класифікації в цьому випадку не будуть працювати.

  В реальному житті такі задачі є в виявленні шахрайських дій з банківськими картами.
- Задача зменшення розмірності – використовуються для зведення великої кількості ознак до меншої (зазвичай 1-3), для зручності подальшої візуалізації. Приклад – стиснення даних.

Загальну кількість завдань, що вирішують за допомогою методів машинного навчання, можна поділити на наступні типи:

- Навчання з вчителем – для кожного прецеденту задається пара «ситуація, необхідне рішення»;
- Навчання без вчителя – для кожного прецеденту задається тільки «ситуація», потрібно згрупувати об'єкти в кластери, використовуючи дані про схожості об'єктів, або зниження розмірності даних;
- Активне навчання – відрізняється тим, що алгоритм має можливість самостійно призначати наступну досліджувану ситуацію, на якій стане відомий правильна відповідь;
- Навчання з підкріпленням – для кожного прецеденту є пара «ситуація, прийняте рішення». Приклад – генетичний алгоритм, який використовується для вирішення завдань оптимізації та моделювання шляхом випадкового вибору;
- Навчання з частковим залученням вчителя (англ. Semi-supervised learning) – для частини прецедентів задається пара «ситуація, необхідне рішення», для іншої частини – тільки «ситуація».
- Різноманітне навчання (англ. Multiple-instance learning) – навчання, коли прецеденти можуть бути об'єднані в групи, для всіх прецедентів є «ситуація», але тільки для одного з них (невідомо якого) є пара «ситуація, необхідне рішення»;
- Багатозадачне навчання (англ. Multi-task learning) – одночасне навчання групи взаємопов'язаних завдань, для кожної з яких задаються індивідуальні пари «ситуація, необхідне рішення».

З огляду на ентропію даних, зробемо висновок, що навчання з вчителем не можливе з наступних причин:

- Неможливо точно детермінувати, які програми будуть виконуватись;
- Неможливо точно сказати, що наведені дані будуть відноситись до "гарного" або "поганого" випадку, через складність відрізнити користувацьку діяльність від злонамірної.

Отже одразу можна відкинути клас алгоритмів, який використовує навчання тільки з вчителем. Оптимальним вибором буде вібир алгоритмів, які базуються на навчані без вчителя.

Дивлячись на різні типи задач, видно, що алгоритми класифікації також не підходять, бо вони базуються на зазделегідь відомих вибірках даних, так само як і алгоритми регресії. Алгоритми виявлення аномалій будуть вказувати на дуже активні процеси (наприклад бази даних або сервери з великим навантаженям) або дуже "неактивні" (наприклад відкритий текстовий редактор vi). Через велику кількість параметрів можна використати алгоритми зменшення розмірності за принципом відбору ознак або виділення ознак.

Увагу слід превернути до алгоритмів кластеризації. Ці висновки зроблені з наступних причин:

- Треба працювати з невизначиними данними: не відомо де користувацька діяльність, а де злонамірна;
- Треба виділяти групи процесів, які належать до однієї або декількох груп та мають спільні риси, а не мати на виході числовий результат, тощо;
- Треба обробляти великий обсяг даних за порівняно малий час;

Скористаемось джерелами [13], [14], [15] та роздивимось детально типи алгоритмів кластеризації та приклади.

Алгоритми квадратичної помилки:

Розв’язок можна розглядати як побудову оптимального розбиття об'єктів на групи. При цьому оптимальність може бути визначена як вимога мінімізації середньоквадратичної помилки розбиття (2.1).

$$e^2(X, L) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} ||x_i^{(j)} - c_j||^2 \qquad (2.1)$$

де cj - «центр мас» кластера j

Найпоширенішим алгоритмом цієї категорії є метод k-середніх. Цей алгоритм будує задане число кластерів, розташованих якнайдалі один від одного. Робота алгоритму ділиться на декілька етапів:

1. Випадково вибрати k точок, які є початковими «центрами мас» кластерів.
2. Віднести кожен об'єкт до кластеру з найближчим «центром мас».
3. Перерахувати «центри мас» кластерів відповідно до їх складу.
4. Якщо критерій зупинки алгоритму не задоволений, повернутися до пункту 2.

Критерій зупинки роботи алгоритму зазвичай вибирають мінімальну среднєквадратичну помилку. До недоліків можна віднести зазделегідь відому кількість кластерів.

Нечіткі алгоритми:

Найбільш відомим є алгоритм c-means. Він є модифікацію методу k-середніх але базуеться на формулі нечіткої помилки. Етапи роботи алгоритма:

1. Обрати початкове нечітке розбиття n об'єктів на k кластерів шляхом вибору матриці належності U розміру n на k.
2. Користуючись матрицю U, знайти значення критерію нечіткої помилки (2.2).
3. Перегрупувати об'єкти з метою зменшення значення критерію нечіткої помилки.
4. Повертатися до пункту 2 до тих пір, поки зміни матриці U стануть незначними.

Недоліки: ці алгоритми можуть не підійти, якщо кожен об’єкт треба віднести до одного кластеру або невідомо кількість кластерів.

$$E^2(X,U) = \sum_{i=1}^{N}\sum_{k=1}^{K} U_{ik}||x_i^{(k)} - c_k||^2 \qquad (2.2)$$

де c_k- «центр мас» нечіткого кластера k (2.3)

$$c_k = \sum_{i=1}^{N} U_k x_i \qquad (2.3)$$

Алгоритми кластеризації графів:

Множина об’єктів в таких алгоритмах представлена у вигляді графа: G = (V, E), вершинам якого відповідають об'єкти, а ребра мають вагу рівну відстані між об’єктами. Перевагою графових алгоритмів є простота імпліментації, представлення даних та зв’язків між даними та можливість вдосконалення. Кластеризація графів може, наприклад, використовувати алгоритм побудови мінімального остовного дерева, виділення зв'язних компонент та алгоритм пошарової кластеризації та інші.

Зазначемо, що алгоритми кластеризації графів найбільш підходять до вирішення задачі поставленій в цій роботі. Оскільки вони не потребують попереднього навчання, виділять множену спільної діяльності та будуть використовувати потужну властивість системних данних – залежність процесів від підпроцессів та використовуваних джерел (файлів, сокетів, приладів).

Приклади:

Алгоритм виділення зв'язаних компонент:

В алгоритмі виділення зв'язаних компонент задається вхідний параметр R. Наступним кроком в графі видаляються всі ребра, для яких відстань між вершинами більше за R. Сполученими залишаються тільки найбільш близькі пари об'єктів. Сенс алгоритму полягає в підборі значення R, що лежить в діапазон всіх відстаней, при якому граф розділиться на кілька зв'язаних компонент. Отримані компоненти і є кластери.

Алгоритм побудови мінімального остовного дерева.

Спочатку будується на графі мінімальне остовне дерево, після цього послідовно видаляється ребра з найбільшою вагою. На малюнку (2.8) зображено мінімальне остовне дерево, отримане для дев'яти об'єктів. На представленій ітерації алгоритма має два кластери {ZXY} та {AGKFMU} через

видалення ребра XU з максимальною вагою 64. Далі більший кластер може бути подріблений на кластери {UM} та {KAGF}.

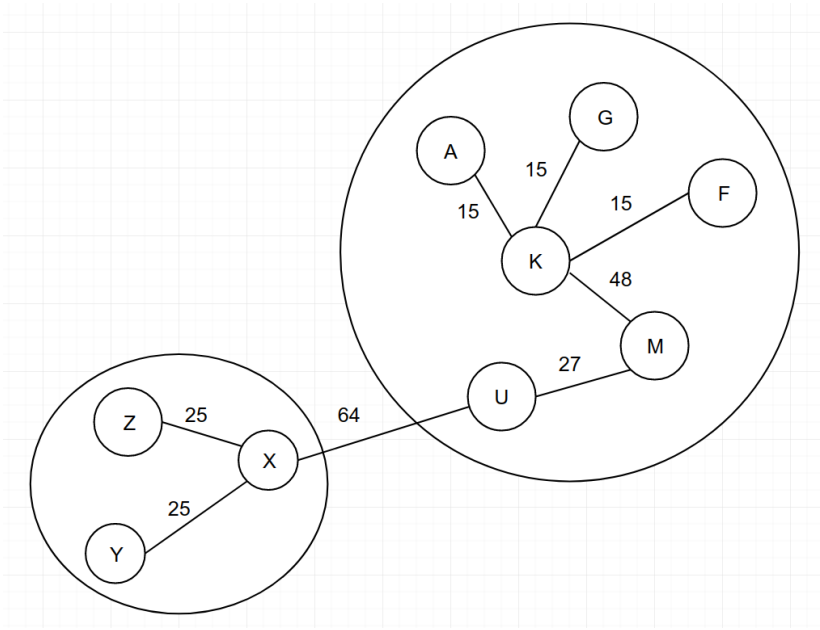

Рисунок 2.8 – Ілютрація алгоритму побудови мінімального остовного дерева

Алгоритм пошарової кластеризації:

Будова базується на зв'язних компонентах з певною відстаню між обєктах. Рівень відстані задається порогом c . Якщо відстань між об'єктами буде наступною: $0 \leq p(x,x') \leq 1$, то c буде мати інтервал від 0 до 1 включно. Даний алгоритм відображає послідовність підграфів графу G, які мають ірархічні зв'язкм між собою (2.0).

$$G^1 \subseteq G^2 \subseteq \cdots G^n \tag{2.4}$$

де $G^t = (V, E^t)$ - граф на рівні c^t, V - сукупність вершин, а E - сукупність ребер, що з'єднують пари цих вершин.

$$E^t = \{e_{ij} \in E: p_{ij} \leq c_t\} \tag{2.5}$$

де c^t – це рівень відстані.

Контролюючи рівень відстані можна задавати рівень кластеризації розділяючи граф як в прощені так і в ірархічному порядку.

Алгоритм ентропії графа (англ. Graph Entropy) [16]:

Цей алгоритм використовує визначення ентропії в сенсу інформаційної теорії. Відомо, що ентропія – це міра невизначеності в якій бере участь випадкова велечина. Застосовується нове визначення: ентропія графу, як міра структурної складності. Алгоритм не потребує ніяких зазделегідь визначених порогів, оскільки він шукає оптимальне рішення шляхом мінімізації ентропії графа. Наведемо етапи:

1. Знайти локально оптимальні кластери з мінімальною ентропією графа. Початкову вершину вибирають навмання.
2. Визначаються сусіди кластера. Сусіди ітеративно оцінюються, мінімізується ентропія зовнішнього графа.
3. Рекурсивно додаються прикордоні вершини, якщо їх додавання спричиняє зменшення ентропії підграфів.

Ентропія вершини v визначається за формулою (2.6):

$$e(v) = -p_i(v) log_2 p_i(v) - p_0(v) log_2 p_0(v) \quad (2.6)$$

де p_i(v) - відношення кількості сусідів v в кластері до загальної кількості сусідів

$$v_i,\ p_0(v) = 1 - p_i(v).$$

Ентропія графа визначається як сума ентропії для всіх вершин графа (2.7):

$$e(G) = \sum_{v \in V} e(v) \quad (2.7)$$

Для зваженого графа ентропія буде приймати значення $p_i(v)$, для вершини v це буде сума вхідних ребер. Зважений ступінь v розраховується за настпною формулою (2.8):

$$p_i(v) = \frac{\sum_{v_i \in V'} w(v, v_i)}{\sum v_k \epsilon V w(v, v_k)} \quad (2.8)$$

Формули залишаються незмінними для методу ентропії зваженого графа.

Алгоритм Coach [17]:

CoAch має 2 етапи:

1. Знаходження найбільш зв'язаних підграфів.
2. Розширення підграфів методом додавання сильно пов'язаних сусідів.

Алгоритм працює з трьома параметрами:

- Мінімальна щільність підграфів;
- Максимальна спорідненість підграфів;
- Мінімальна відстань між сусідами.

Підграфи кандидати з однією вершиною об'єднуються у в глобальний набір попередніх підграфів. Щоб це застосувати використовується «фільтрування надмірності», кожний підграф кандидат оцінюється як кожні попередні підграфи з мірою пов'язаності (2.9) для деякого підграфа A та підграфа B.

$$NA(A,B) = \frac{|V_A| \cap |V_B|}{|V_A| X |V_B|} \tag{2.9}$$

Якщо показник NA для підграфа кандидата та його найбільш аналогічного попереднього підграфа B_m ax, менше ніж визначений користувачем поріг спорідненості підграф додається до набору попередніх підграфів B. В іншому випадку, якщо щільність підграфа кандидата ($d_A \times |V_A|$) більше ніж $B_m$ax, тоді підграф кандидат $B_m$ax переводиться в набір попередніх підграфів. Після виявлення підграфів для кожної вершини, попередні підграфи розширюються на прогнозовані кластери.

Для кожного попереднього підграфа C = (V_c, E_c), CoAch приймає набір безпосередньо сусідніх підграфів, NC. Потім алгоритм додає всі v ∈ NC, тобто більше ніж половини вершин у C. Ця міра також відома як близькість v до C (2.10):

$$closeness(v, C) = \frac{|N_v| \cap |V_c|}{|V_c|} \tag{2.10}$$

де N_v – набір сусідів v.

Вагова версія CoAch використовує зважені функції ступеня та щільності.

Алгоритм DPClu [18]:

DPClus оцінює вагу на графі за допомогою підрахунку сусідів. У DPClus вага дуги (u, v) ∈ E визначається як кількість загальних сусідів між u та v. Аналогічно, вага вершини - це зважений ступінь або сума всіх ребер, з'єднаних з вершиною. Умова зупинки формування кластерів заснована на щільності, а не

на вагах вершинни. DPClus покладається на два елементи зупинки для кластеризації: щільність кластера (d_{in}) та властивість кластера (cp_{in}). Щільність d_k кластера $C_k = (V_k, E_k)$ визначається як відношення кількості ребер у кластері до кількості можливих ребр в кластері (2.11).

$$d_k = \frac{E_k}{|V_k| X (|V_k| - 1)} \tag{2.11}$$

Властивість кластера cp_{nk} між вузлом n та кластером C_k визначається як зважена зворотна щільність C_k (без n) за часткових вершин кластера, які примикають до n (2.12).

$$cp_{nk} = \frac{|N_n \cap V_k|}{d_k X |V_k|} \tag{2.12}$$

де N_n - сукупність вершин, що примикають до n.

Для вибору центра обирається вершина з найбільшою вагою, якщо тільки найвища вага не дорівнює нулю, у цьому випадку вершина з найвищим ступенем вибирається як центр. Кластер вирощується ітераційно, вибір вершини з найвищим пріорітетом робиться з групи сусідів. Пріорітет сусідської вершини визначається наступним чином:

1. Кількість вузлів кластера, що з'єднані з сусідом;
2. Сума ваги ребем між сусідом та його суміжними вузлами.

Зростання кластера припиняється коли властивості кластера падають нижче відповідного порогу. Після цього всі вузли в кластері видяляються та ваги вершин отриманого графа перераховуються. Алгоритм припеняється коли в графі не залишається ребер.

Алгоритм IPCA [19]:

IPCA це модифікація алгоритму DPClus. На відміну від попереднього цей метод фокусується на підтримці діаметру кластера, який визначається як максимально коротка відстань між усіма парами вершин. Алгоритм обчислює локальну вершину та зважені ребра, підраховуючи кількість сусідів розподілених між парами вершин. Ці дані розраховуються один раз на початку алгоритму. Це

дозволяє природнім чином створювати перекриття між кластерами, оскільки вершини в кластері не існують постійно. Слід зазначити, що даний підхід може призвести до великої кількості перекриття кластерів. Алгоритм може використовувати два порядки оцінки сусіда: (SP ≤ d) та (ASP ≤ d). Виходячи з (SP ≤ d) потрібно, щоб діаметр кластера ніколи не перевищував d, а (ASP ≤ d) враховує середню довжину найкоротших шляхів між усіма парами вершин в кластері. Вибір початкової вершини здійснюється так само як і в DPClus. Кластер K поширюється, обираючи своїх сусідів за значенням IN_{vK} яке визначається за формулою (2.13).

$$IN_{vK} = \frac{m_{vK}}{|K|} \tag{2.13}$$

де m_{vK} – кількість ребер між вершиною v та кластером.

Вершина буде додана до кластеру, якщо вона відповідає двом умовам:

1) $IN_{vK} \geq T_{in}$;
2) SP (K + v) ≤ d.

T_{in} – поріг, визначений користувачем мінімальний відсоток сусідніх кластерних вершин, з якими v має зв'язок, щоб бути доданим до кластеру. Друга умова вимагає максимально короткої довжини шляху між v і кожним членом K, вона не повинна бути більшою ніж d. Як правило, d встановлюється як два. Для інтеграції вагів ребер в IPCA значення IN_{vK} задається формулою (2.14).

$$IN_{vK} = \frac{\sum_{u \epsilon K} w(u, v)}{|K|} \tag{2.14}$$

Вимога IPCA, що $IN_{vK} \geq T_{in}$ є набагато значною у зваженому графі. Відзначимо, якщо вершина обрана як початкови вузол та не буде зв'язена глобальними ребрами ваги яких, більше за T_{in}, жодні сусіди не будуть додані до кластеру. У цьому випадку обрана вершина позначенна як відвідана, але про кластер розміру n-1 не повідомляється, як результат. Якщо сусідня вершина відповідає обом зазначеним вище критеріям, віна додається до кластеру та пріорітет сусідів нового кластера перераховуються. Якщооптимального сусіда немає знайдено, зростання кластера припиняється, а кластер виводиться як

результат. Вершини в результуючому кластер позначаються як відвідувані, вони все ще можуть бути включені до майбутніх кластерів але не можуть бути обраними для побудови кластера.

Висновки до розділу 2

Проаналізовані проблеми виявлення діяльності користувачів в системі. Серед них виділяються найбільш важливі:

1) Різноманітність представлення даних та самих даних, залежність зміни за часом;
2) Відсутність будь-якої детермінованості даних;
3) Складність відрізнити діяльність операційної системи та користувача від діяльності зловмисника;
4) Різноманітність параметрів та залежностей в данних;
5) Складність в обробці через велиці обсяги.

Беручи до уваги міркування наведені в підрозділі 2.3 зробимо висновок, що попередньо згадані проблеми можна вирішити за допомогою методів машиного навчання, а саме методами кластеризації без вчителя, що надає можливість розробити рішення не залежне від програмного оточнення та діяльності користувача.

Запропонованно дві моделі представлення даних для відображення діяльність користувачів в системі:

- Відображення множини процессів в множену n-мірного простору, де для кожно процесу буде існувати відповідна точка.
- Відображення множени процессів в граф, де для кожного процесу або інформаційного об’єкту буде існувати відповідна вершина. Вага ребер розраховується з часу протягом якого існував зв’язок між вершинами.

Слід зазначити, що дуже великою особливістю проаналізованих даних є зв’язки між елементами виду n до n. Ця особливість закладена в архітектурі операційних систем. Отже для подальшої роботи найбільш вдалим буде вибір алгоритмів кластеризації графів:

- Graph Entropy;
- Coach;
- DPClu;
- IPCA.

Виходячи з їх всластивості виділяти кластери з великою щільністтю зв’язків та вагових коефіцієнтів.

3 РЕАЛІЗАЦІЯ ВИЯВЛЕННЯ APT-АТАК НА ОСНОВІ АЛГОРИТМІВ КЛАСТЕРИЗАЦІЇ ГРАФІВ

В даному розділі проведемо експериментальні дослідження на основі інформації, наведеної у попередніх розділах. Розробимо відповідний програмний комплекс. Змоделюємо поведінку звичайного адміністратора та поведінку зловмисника в рамках однієї операційної системи. Застосуємо засоби збору інформації з ядра операційної системи.

Після експериментального збору інформації, обробемо її за допомогою алгоритмів кластеризації графів. Проаналізуємо результати.

3.1 Побудова експериментального програмного комплексу

Формально програмний комплекс ділиться на чотири составні частини: емуляція поведінки користувачів, збір інформації, обробка та аналіз інформації, представлення результатів.

Емуляція поведінки:

Робота адміністратора буде емулюватись но основі повсякденних загальних завдань: моніторінг системи, робота з файловим менеджером, використання засобів віддаленого керування, робота з програмним кодом, пошук інформації в інтернеті, віртуалізація.

Злонамірна діяльність буде емулюватись на основі рис APT-атак описаних в розділі 1.2. Для цього в операційну систему навмисно вбудовані відомі бекдори та вразливі сервіси. Також створенно відповідне мережеве оточення з вразливими хостами, php сервером, sql базою даних для емуляції горізонтального мережевого руху та пошуку інформації.

Використовувана операційна система — 86_64 GNU/Linux 16.04.1-Ubuntu. Версія ядра — 4.15.0-66-generic.

Збір інформації та обробка:

Для вирішення даної задачі використовується утіліта lsof [20]. lsof дає інформацію про файли, відкриті процесами системи для наступних діалектів UNIX:

- AIX 5.3
- Apple Darwin 9 (Mac OS X 10.5)
- FreeBSD 4.9 for x86-based systems
- FreeBSD 7.[012] and 8.0 for AMD64-based systems
- Linux 2.1.72 and above for x86-based systems
- Solaris 9 and 10

Після обробки дані заносяться в базу даних MongoDB, злонамірна діяльність маркується для подальшого порівняння результатів.

Аналіз та обробка результатів:

На цьому етапі відбувається аналіз зібраної інформації в графовому вигляді за допомогою бібліотеки з методами кластеризації [21], а саме: Graph Entropy, Coach, DPClu, IPCA. Отримавши результати для кожного алгоритму визначаються результати в кількісних характеристиках, обираються кластери з найбільшою кількісттю інформації про злонамірну активність, відповідно маркуються та заносяться в базу даних.

Графічне представлення результатів:

На виході програмного комплексу отримаємо кількісні характеристики результатів роботи алгоритмів кластеризації графів, а також графічне представлення [9]: графи з усією діяльністю, підграфи-кластери з знайденою злонамірною діяльністю та підграфи з попередньо відомою злонамірною діяльністю.

Проміжні обчислення та кластеризація графів проводились з допомогою мови python, збір системної інформації та моделювання графів з допомогою мов C та C++.

Схематично архітектура та потоки інформації програмного комплексу зображені на рис. 3.1.

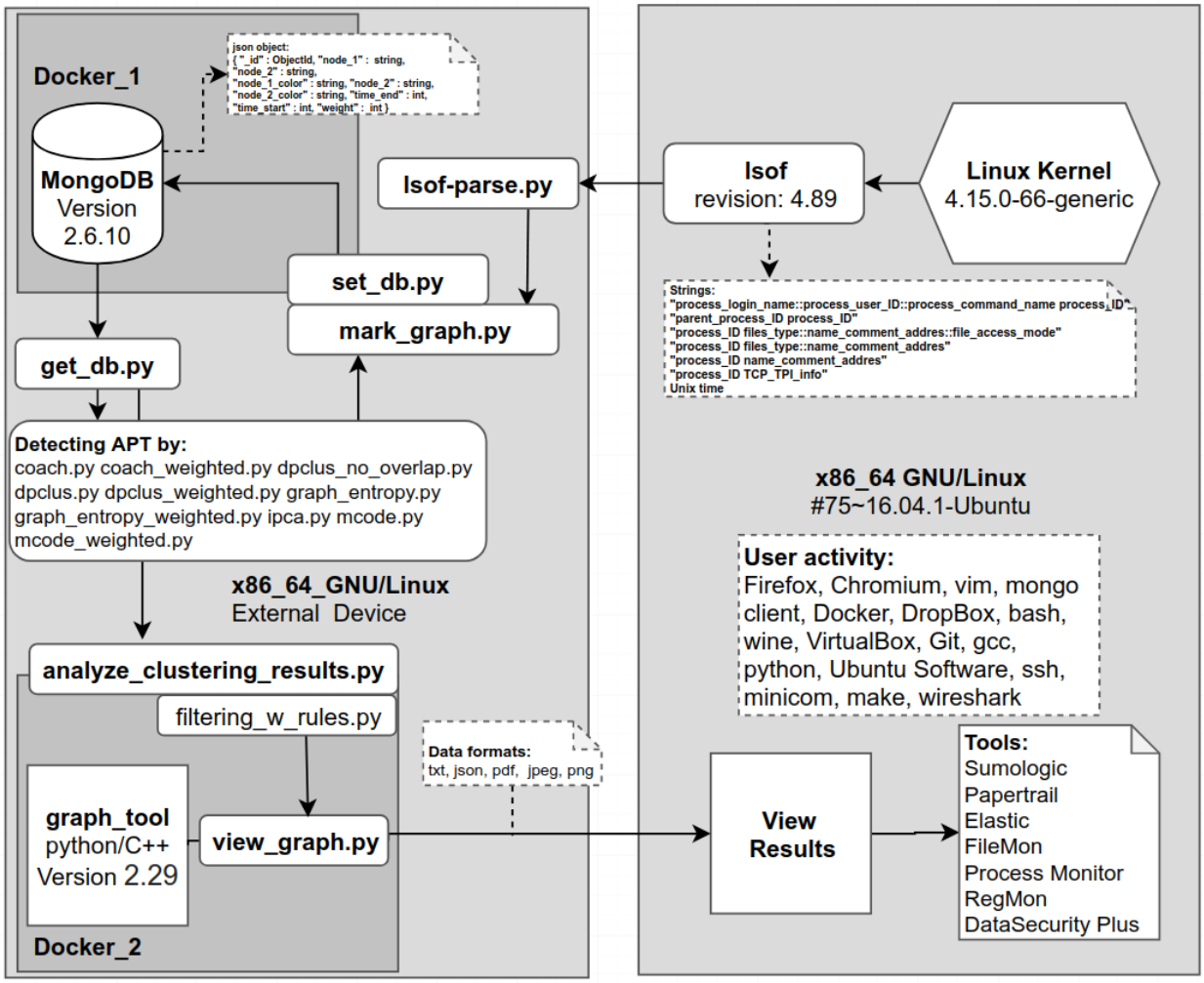

Рисунок 3.1 – Схема розробленого програмного комплексу

3.2 Емуляція поведінки системних користувачів

Як зазначалось в попередньому розділі, в ході експеримента буде приймати участь 3 ролі: адміністратор система та зловмисник.

Цікавити буде саме поведінка зловмисника, вона буде максимально наближеною до моделей описаних у розділі 1.2, тобто буде відтворенно проникнення в операційну систему, через вразливі клієнти, використання свіжої вразливості програми sudo [22] для підвищення привілєїв, встановлення

додаткового програмного забезпечення, мережева розвідка та збір приватної інформації.

Перелік перелік програм, які використані зловмисником: nmap, sqlmap, mongo, nslookup, tcpdump, arp, ping, tracepath, tftp, ssh, wget, netstat, lsof, cfg80211, su, sudo, python3, zsh, sh, htop, grep, vi, find, ls, cd, bash, tar, gzip.

Перелік програм, які використані адміністратором: docker, lightdm, lsof, ssh, sudo, zsh, login, chromium-browse, python3, nautilus, mongo, dropbox, dropbox.bash, subl, sublime_text, Media, git, git-gui, apt-get, firefox, top, htop, vim, /etc/init.d/*, tftp, wine, gcc, make, terminator, wireshark.

Стислий перелік програм запущених автоматично системою: syslog, system, uuidd, whoopsie, gdbus, gmain, wpa_supplicant, Xorg, watchdog, winbindd, upowerd, smbd, snapd, SignalSender, kworker, ksoftirqd, bluetoothd, cpuhp, accounts-daemon, sshd, NetworkManager, gfx, docker-proxy, cups-browsed, crypto, slapd, dnsmasq, bus-daemon.

Зазначемо, що деякі програми можуть запускати інші підпрограми, які не були попередньо описані.

3.3 Збір та обробка інформації про системну діяльність

Для збору інформації використовується утіліта lsof [20], з наступними параметрами “-FacCdDfFGiklLmnNopgPrRsStTuzZ -r1”. Наведемо значення флагів виводу інформації:

- L – process_login_name: ім’я користувача запустившого процес;
- u – process_user_ID: унікальний ідентифікатор користувача;
- c – process_command_name: назва програми;
- p – process_ID: унікальний ідентифікатор процесу;
- R – parent_process_ID: унікальний ідентифікатор батьківського процесу;

- t – files_type: тип відкритого файлу;
- n – name_comment_addres: ім’я файлу, коментарії або мережева адреса;
- a – file_access_mode: режим доступу до файлу (r, w, u);
- T – TCP_TPI_info: інформація що до мережевого з’єднання.

На виході lsof отримаємо інформацію про процеси в зручному вигляді для парсингу див. рис 3.2. Інформація збирається кожну секунду.

```
f128aul tunixG0x802;0x0d0xffff932787ef9000o0t0i59158n@/tmp/.X11-unix/X0 type=STREAM
f129aul tREGG0x8002;0x0D0x5s4i66298k0n/memfd:xshmfence (deleted)
f130aul tunixG0x802;0x0d0xffff932787efb400o0t0i63649n@/tmp/.X11-unix/X0 type=STREAM
f131aul tREGG0x8002;0x0D0x5s4i59170k0n/memfd:xshmfence (deleted)
f132aul tunixG0x802;0x0d0xffff9328b77e5400o0t0i63653n@/tmp/.X11-unix/X0 type=STREAM
f133aul tREGG0x8002;0x0D0x5s4i90632k0n/memfd:xshmfence (deleted)
f134aul tREGG0x8002;0x0D0x5s4i74469k0n/memfd:xshmfence (deleted)
f135aul tREGG0x8002;0x0D0x5s4i146283k0n/memfd:xshmfence (deleted)
f136aul tREGG0x8002;0x0D0x5s4i146280k0n/memfd:xshmfence (deleted)
f137aul tREGG0x8002;0x0D0x5s4i67000k0n/memfd:xshmfence (deleted)
f138aul tunixG0x802;0x0d0xffff932770fdf800o0t0i93484n@/tmp/.X11-unix/X0 type=STREAM
f139aul tREGG0x8002;0x0D0x5s4i94361k0n/memfd:xshmfence (deleted)
f140aul tREGG0x8002;0x0D0x1as4i349k0n/dev/shm/shmfd-N501nB (deleted)
f141aul tREGG0x8002;0x0D0x5s4i94359k0n/memfd:xshmfence (deleted)
f142aul tREGG0x8002;0x0D0x5s4i94360k0n/memfd:xshmfence (deleted)
p1252g1252R1086cInputThreadu0Lroot
fcwda l tDIRD0x10301s4096i2k25n/
frtda l tDIRD0x10301s4096i2k25n/
ftxta l tREGD0x10301s2419848i2230515k1n/usr/lib/xorg/Xorg
fDELa l tREGD0x19i2514n/i915
fDELa l tREGD0x19i43n/i915
fDELa l tREGD0x19i467n/i915
fDELa l tREGD0x19i1820n/i915
fDELa l tREGD0x5i8454162n/SYSV00000000
fmema l tCHRD0x6r0xe200i323k1n/dev/dri/card0
fDELa l tREGD0x19i542n/i915
fDELa l tREGD0x5i2490384n/SYSV00000000
fDELa l tREGD0x5i2031631n/SYSV00000000
fDELa l tREGD0x5i1310728n/SYSV00000000
fDELa l tREGD0x19i2391n/i915
fDELa l tREGD0x19i2377n/i915
fDELa l tREGD0x19i2196n/i915
fDELa l tREGD0x19i1885n/i915
fDELa l tREGD0x5i9404441n/SYSV00000000
fDELa l tREGD0x19i2282n/i915
fDELa l tREGD0x19i2174n/i915
```

Рисунок 3.2 – Результат роботи lsof

Результат роботи lsof парситься python скриптом lsof-parse.py. після парсингу скрипт формує ребра майбутнього графу згідно зазначеній на рис. 3.3 структурі дани, сортуе відповідно алфавіту та додає часову відмітку існування зв’язку. Для більш детального представлення зверніться до додатку Б. Наведену структуру даних використовують алгоритми кластеризації даних. Слід зазначити, що вага ребера визначається часом існування зв’язка між вершинами.

```
Struct format: "NODE_1 NODE_2 UNIXTIME".

Note: 1) process_ID, TCP_TPI_info or files_type are types of specific strings.
      2) "::" used for concatenate strings.

Exampe:
"process_login_name::process_user_ID::process_command_name"  "process_ID" UnixTime
"parent_process_ID"                                          "process_ID" Unixtime
"process_ID"          "files_type::name_comment_addres::file_access_mode" Unixtime
"process_ID"                          "files_type::name_comment_addres" Unixtime
"process_ID"                                    "name_comment_addres" Unixtime
"process_ID"                                           "TCP_TPI_info" Unixtime
```

Рисунок 3.2 – Структура даних для аналізу агоритмами кластеризації графів

Наступним кроком буде маркування. Усі вершини майбутнього графу позначаються як “сірі” окрім вершин, які відповідають злонамірній діяльності, вони в свою чергу позначаються як “помаранчеві”.

Оброблена інформація заноситься в базу даних як json об’єкт в вигляді наведеному на рис 3.4. Поле “weight” рахується як різниця полів “time_start” та “time_end”.

```
json object:
{
        "_id" :          ObjectId,
        "node_1" :       string,
        "node_2" :       string,
        "node_1_color" : string,
        "node_2" :       string,
        "node_2_color" : string,
        "time_end" :     int,
        "time_start" :   int,
        "weight" :       int
}
```

Рисунок 3.3 – Структура даних для бази даних

Наведемо приклади даних у наведеному форматі для алгоритмів кластеризації про злонамірну активність на рис. 3.4. На рисунку можно побачити підвищення привілеїв, мережеве сканування та відповідні використані джерела.

```
j3::1000::sh 3304 195
2360 3304 66
j3::1000::sudo 3315 195
root::0::bash 10215 191
10752 /lib/x86_64-linux-gnu/ld-2.23.so 3
10752 /lib/x86_64-linux-gnu/libc-2.23.so 3
10294 /usr/bin/nmap 2
10294 /usr/lib/libblas/libblas.so.3.6.0 2
root::0::python3 10177 194
10177 /usr/bin/python3.5 194
10177 /usr/lib/locale/locale-archive 194
10177 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache 194
root::0::snapd 1020 5070
root::0::sqlmap 10358 74
10358 /lib/x86_64-linux-gnu/libz.so.1.2.8 74
10358 pipe 148
10358 /proc/10358/task/10364/fd/8_(readlink:_No_such_file_or_directory) 1
10358 protocol:_TCP 108
10358 QS=0 270
10358 QS=1 6
10358 /usr/bin/python2.7 74
10358 /usr/lib/locale/locale-archive 74
root::0::ssh 10439 30
10439 j3-ThinkPad-E480:36428->192.168.43.248:ssh 30
10439 /lib/x86_64-linux-gnu/ld-2.23.so 30
```

Рисунок 3.4 – Приклад злонамірної діяльності

В ході експеримента було зібранно п’ять датасетів з відповідною назвою data_1, data_2, data_3, data_4, data_5 з різною поведінкою користувачів загальним обсягом 2.4 гігабайтів. Загальний час збору даних дві години. Після сортування та обробки даних, для представлення в графовому вигляді розмір даних значно зменшився див. таблицю 3.1. Зменшення відбулося через групування дублікатних зв’язків та процесів зібраних протягом часу.

Таблиця 3.1 – Загальні характеристики зібраних даних.

Dataset name	data_1	data_2	data_3	data_4	data_5	**Total**
Size	702K	788K	1,2M	699K	578K	**3,9M**
Vertex	2764	3257	4668	3237	2498	**16424**
Edge	14844	16505	23455	14871	12430	**82105**

З таблиці видно, що обсяг даних зменшився з 2.4 гігабайта до 3.9 мегабайта. Також на кожну вершину в середньому приходиться до п’яти зв’язків.

Наведемо для кожного алгоритма кластерізації графів час роботи в таблиці 3.2; обсяг даних 1,2M.

Таблиця 3.2 – Час роботи алгоритмів кластеризації графів.

Algorithm	coach	coach_weighted	dpclus	dpclus_weighted
Time	0,22s	0,42s	169,23s	91,15s
Algorithm	graph_entropy	graph_entropy_weighted	ipca	
Time	59,55s	273,03s	42,30s	

3.4 Аналіз отриманих результатів, визначення ефективних алгоритмів

Для експерементальної реалізації попередньо відомо які процесси відносіться до злонамірної діяльності. Виділити потрібний кластер не складно. Що до прикладної реалізації, слід вводити додаткову фільтрацію кластерів за індивідуально налаштованими під операційну систему та користувачів правилами для визначення процеса як частини злонамірної діяльності.

Приклади правил:

- Перевірка назви батьківського процеса для визначених програм;
- Перевірка директорії виконання для визначених програм;
- Перевірка назв використовуваних бібліотек для визначених програм;
- виклик недозволених програм з командної оболонки для визначених користувачів;
- відкриття заборонених мережевих з'єднань для визначених користувачів;
- велика кількість відкритих файлів на читання або запис інформації для визначених програм;
- виклик скриптів з попередньо невідомою назвою;

- доступ до конфігураційних файлів;
- доступ до файлів розподілу прав;

Приклади злонамірної діяльності в даних див. на рис 3.4.

Отже для прикладної реалізації будуть обиратися найбільші кластери для яких спрацювала переважна кількість правил. Виходячи з поставленої задачі – треба знайти кластери з максимальною кількістю процесів злонамірної діяльності, та мінімальною кількістю сміття (процесів користувацької або системної діяльності), оберемо наступні показники:

- Процент кількості знайдених злонамірних процесів в кластері;
- відношення злонамірних процесів до загальної кількості процесів – відносна однорідність;
- Інформативність кластера, яка обчислюється за формулою 3.1.

$$I_i = \frac{n_i}{N} * \frac{K_i - c_i}{K_i} * 100 \qquad (3.1)$$

Де I_i – інформативність кластера i, $\frac{n_i}{N} * 100$ – процент знайдених злонамірних процесів, $\frac{K_i - c_i}{K_i}$ – відношення злонамірних процесів до загальної кількості (далі позначається як однорідність) в кластері, n_i – кількість знайдених злонамірних процесів, c_i – користувацькі та системні процеси, N – загальна кількість злонамірних процессів, K_i – загальна кількість процесів в кластері i.

Використовуючи наведену формулу, проаналізуємо кількість процесів в кожному кластері та підрахуємо інформативність кластерів для наступних алгоритмів: coach, coach_weighted, dpclus, dpclus_weighted, graph_entropy, graph_entropy_weighted, ipca. В результатах не будуть приймати участь кластери в яких кількість злонамірних процесів дорівнює нулю. Аналіз проводиться для кожного з п’яти датасетів. Наведемо результати у вигляді таблиць та діаграм (тільки для перших двох датасетів, через великий обсяг інформації).

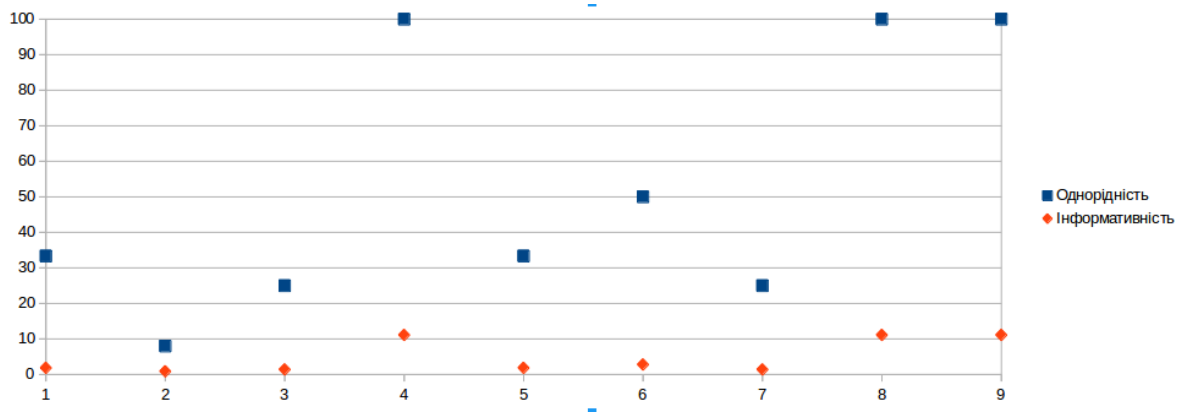

Рисунок 3.5 – Результат алгоритму кластеризації coach для data_1

Рисунок 3.6 – Результат алгоритму кластеризації coach_weighted для data_1

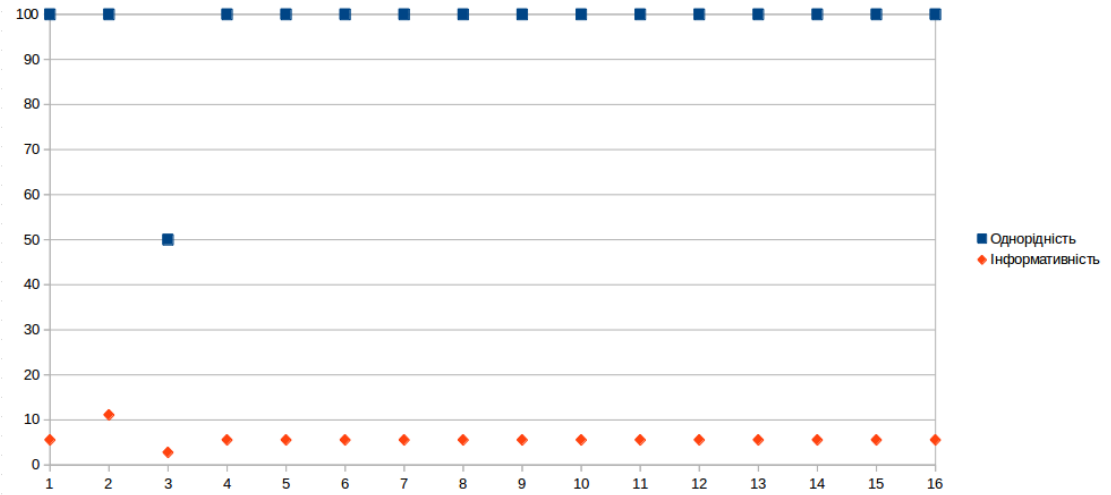

Рисунок 3.7 – Результат алгоритму кластеризації dpclus для data_1

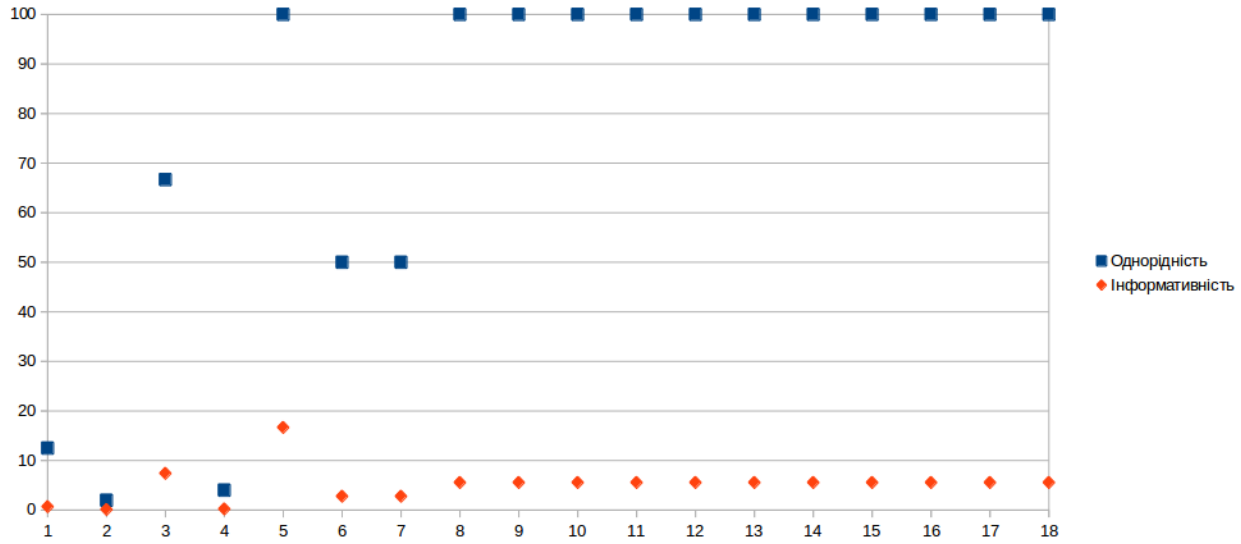

Рисунок 3.8 – Результат алгоритму кластеризації dpclus_weighted для data_1

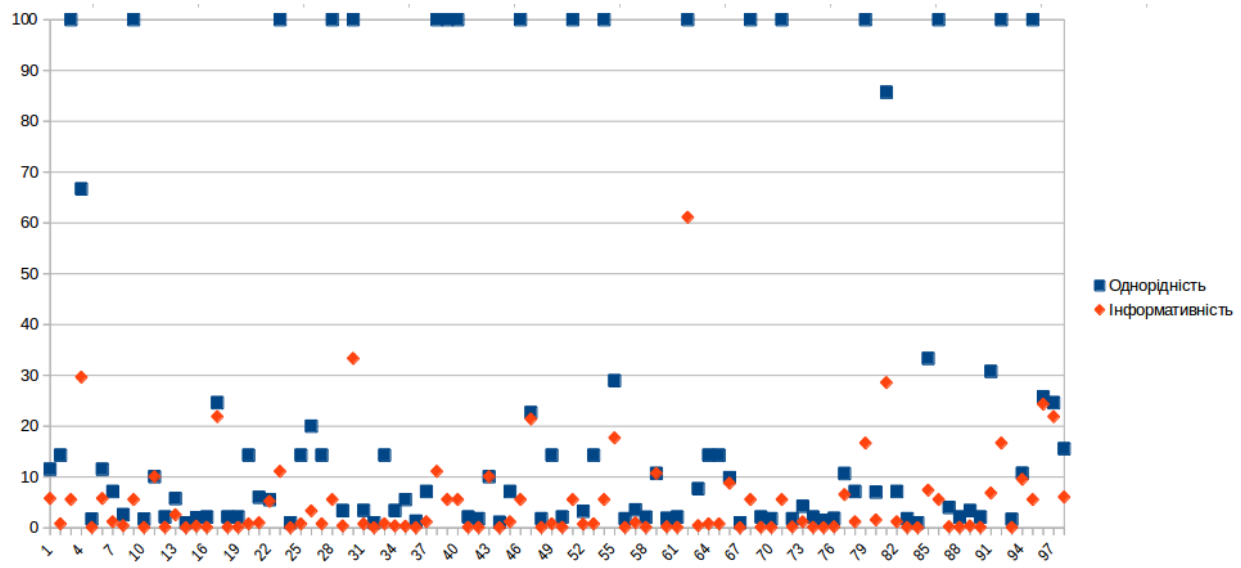

Рисунок 3.9 – Результат алгоритму кластеризації graph_entropy для data_1

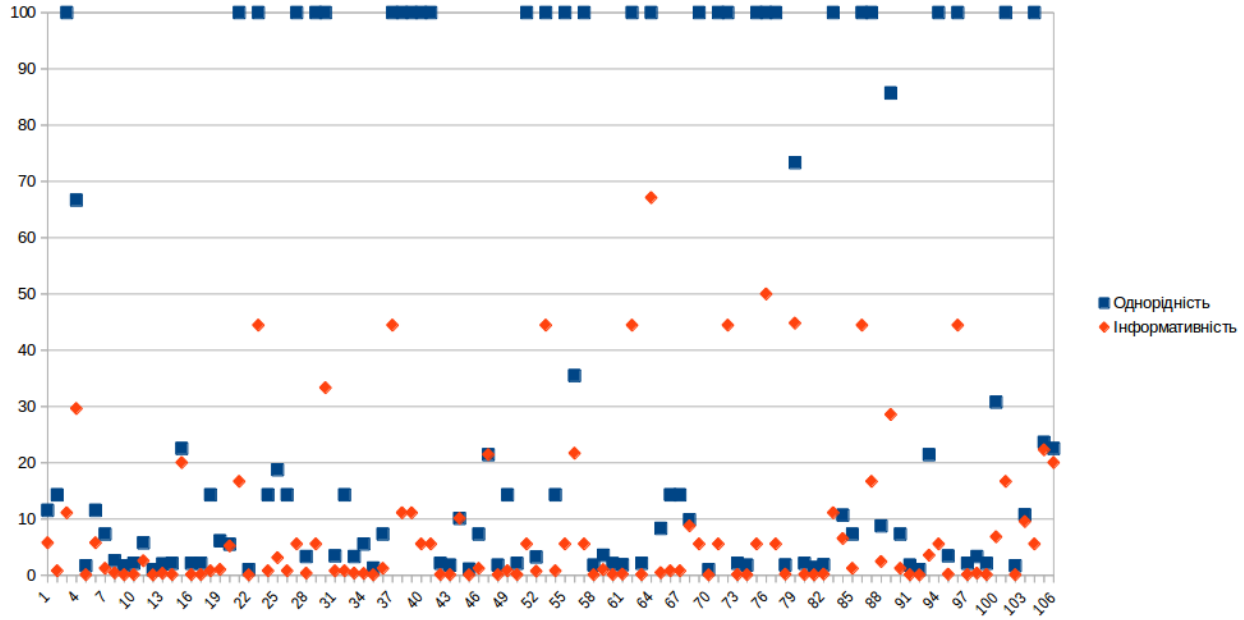

Рисунок 3.10 – Результат алгоритму кластеризації graph_entropy_weighted для data_1

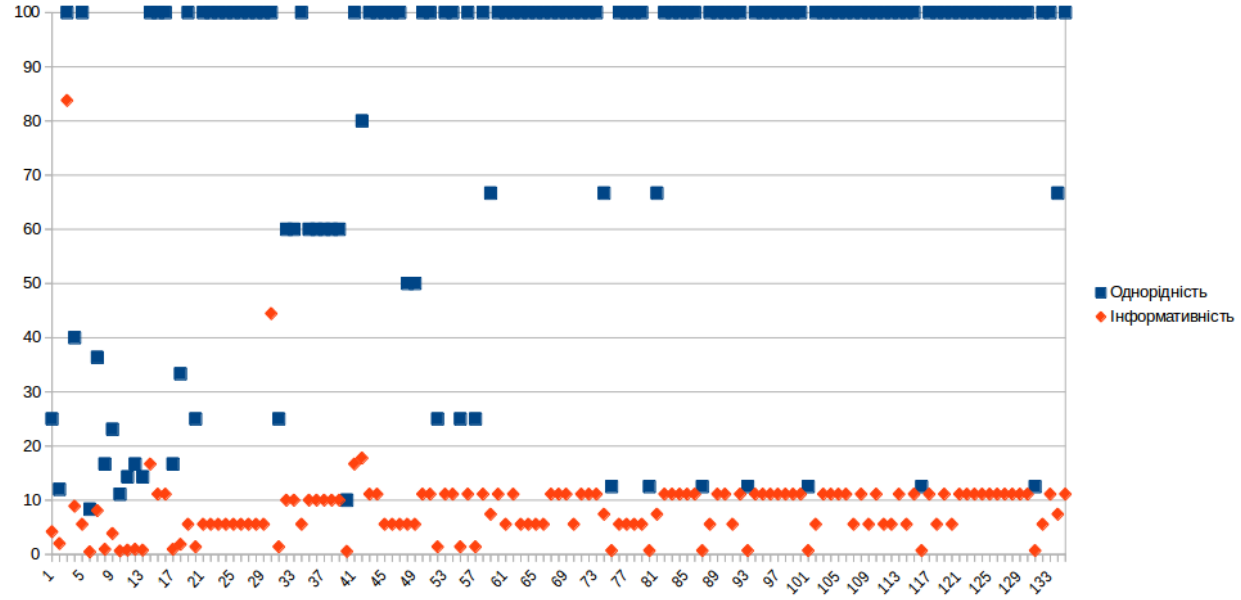

Рисунок 3.11 – Результат алгоритму кластеризації ipca для data_1

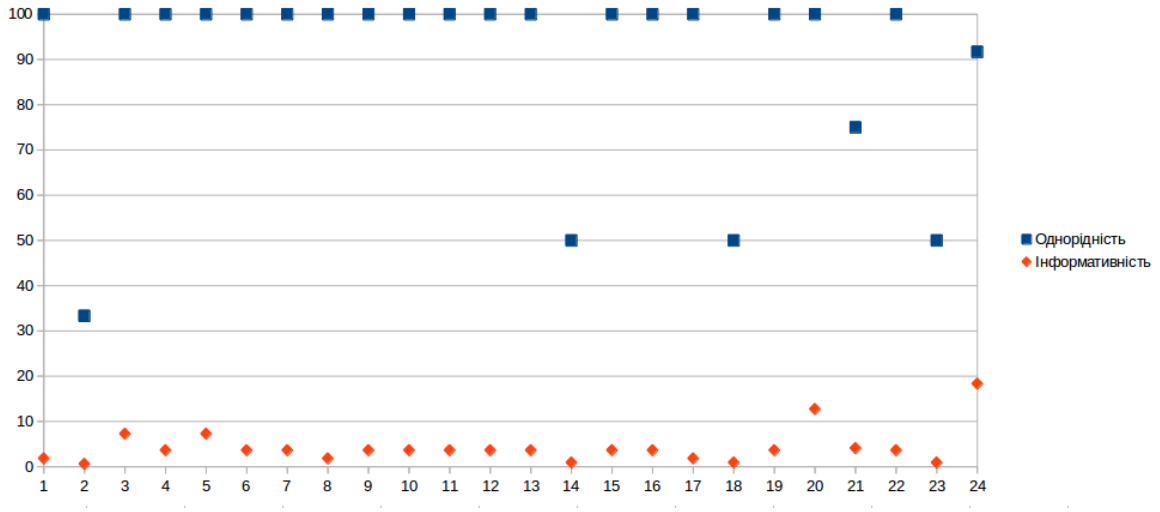

Рисунок 3.12 – Результат алгоритму кластеризації coach для data_2

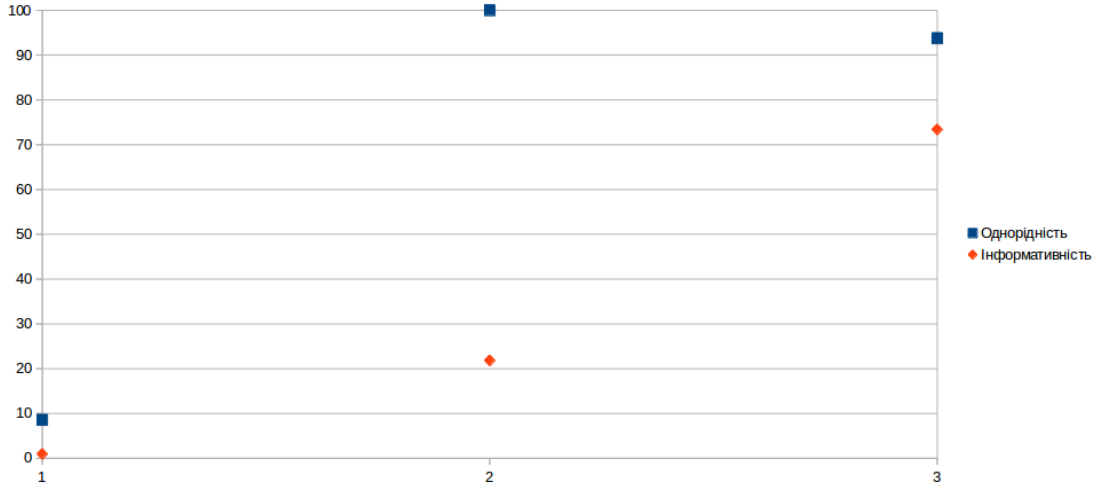

Рисунок 3.13 – Результат алгоритму кластеризації coach_weighted для data_2

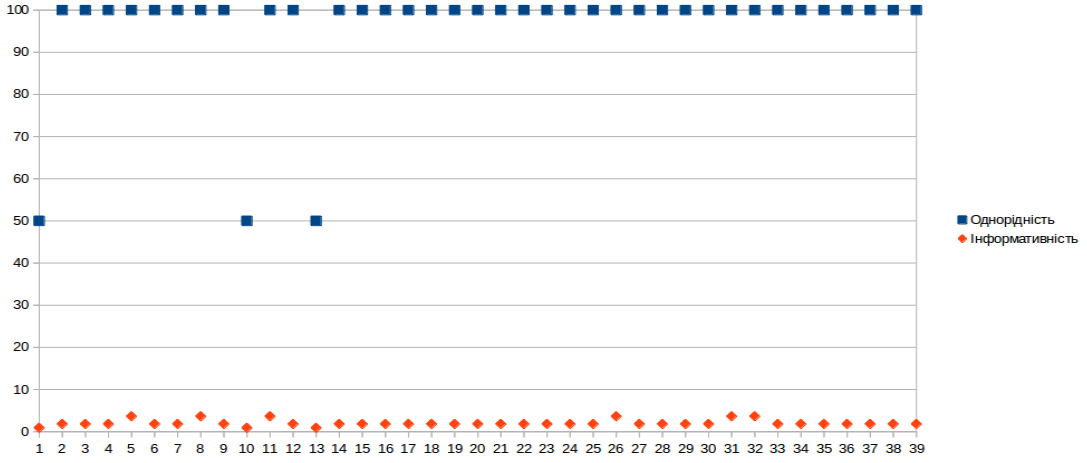

Рисунок 3.14 – Результат алгоритму кластеризації dpclus для data_2

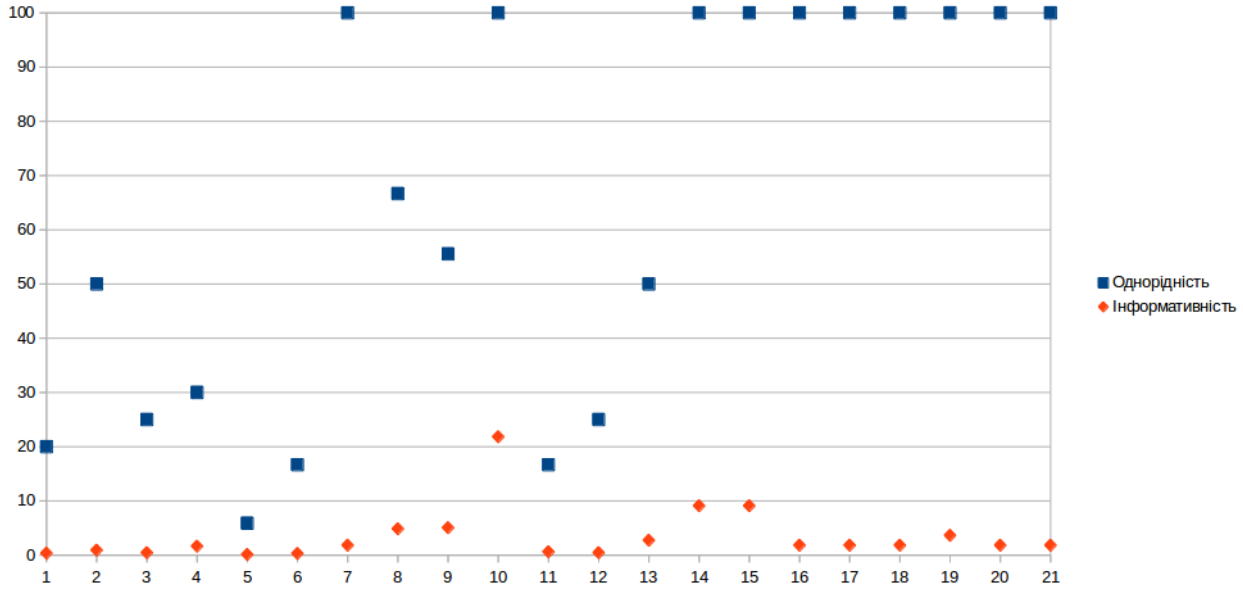

Рисунок 3.15 – Результат алгоритму кластеризації dpclus_weighted для data_2

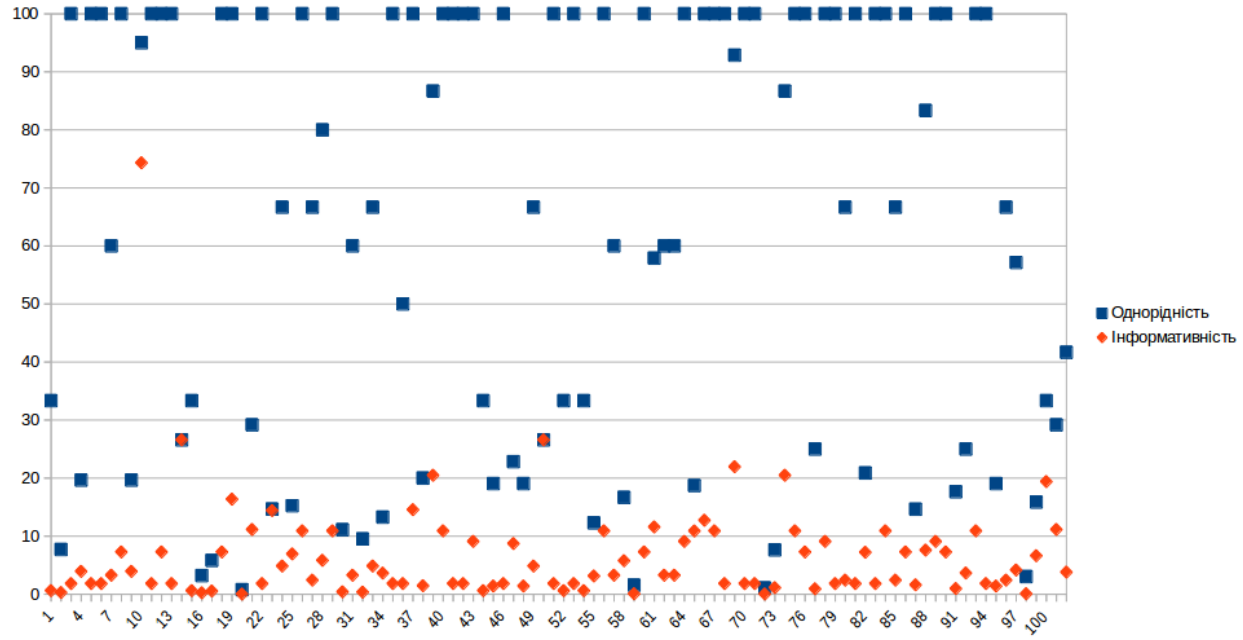

Рисунок 3.16 – Результат алгоритму кластеризації graph_entropy для data_2

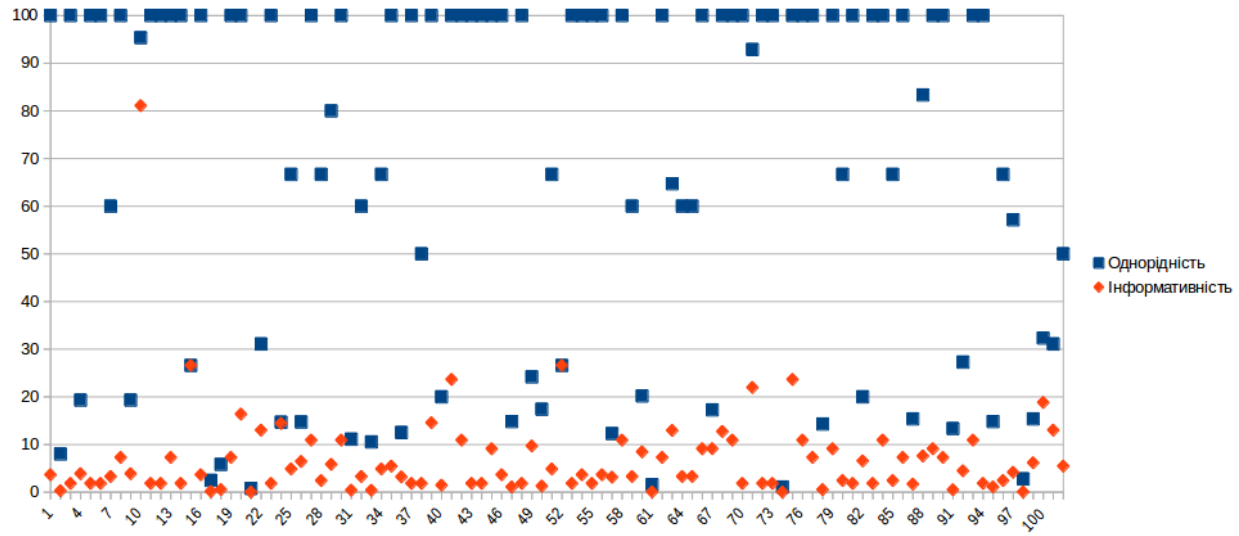

Рисунок 3.17 – Результат алгоритму кластеризації graph_entropy_weighted для data_2

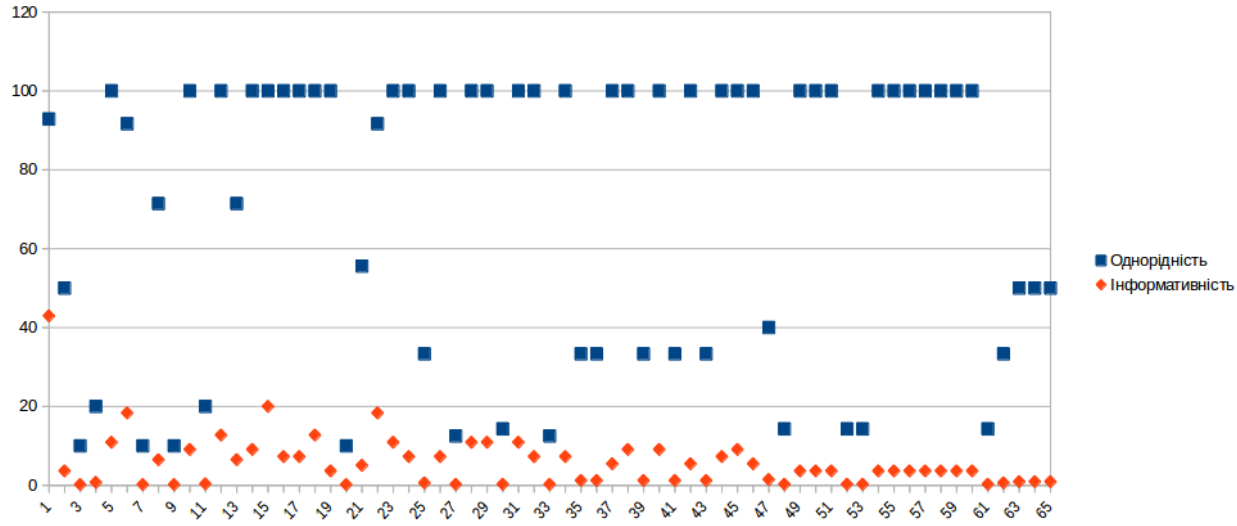

Рисунок 3.18 – Результат алгоритму кластеризації ipca для data_2

Обчислемо для кожного алгоритму та для кожного датасету (data_1, data_2, data_3, data_4, data_5) максимальну інформативність, підрахуємо мінімум, максимум та середнє значення; результати занесемо у таблицю 3.3 та таблицю 3.4.

Оцінивши наглядно наведені діаграми, та дані наведені в таблицях можна впевнено сказати, що алгоритми coach, dpclus та dpclus_weighted не бажанно використовувати в рішеннях подібних задач.

Таблиця 3.3 – Максимальні значення інформативності кластера відповідно до датасетів та алгоритмів кластеризації.

	data_1	**data_2**	**data_3**	**data_4**	**data_5**
coach	11,11	18,33	18.75	21,05	10
coach_weighted	68,06	73,37	45.14	66,68	63,47
dpclus	12,68	3,64	6,25	10,53	12,61
dpclus_weighted	16,67	21,82	7,03	21,93	8,89
graph_entropy	61,11	74,31	61,08	68	78,03
graph_entropy_weighted	67,11	81,08	73,68	72,31	84,09
ipca	83,78	42,95	54,09	72,11	46

Таблиця 3.4 – Максимальні, мінімальні та середні значення інформативності кластерів відповідно до алгоритмів кластеризації.

	Min	**Max**	**Average**
coach	10	21,05	15,1225
coach_weighted	45,14	73,37	67,895
dpclus	3,64	12,61	9,142
dpclus_weighted	7,03	21,93	15,268
graph_entropy	61,08	78,03	68,506
graph_entropy_weighted	67,11	84,09	75,654
ipca	42,95	72,11	59,786

Найкраще себе проявили алгоритми coach_weighted, graph_entropy, graph_entropy_weighted для якого максимальні значення перевищують 70% інформативності. Серед обраних алгоритмів graph_entropy_weighted має найбільші максимальні та мінімальні значення – його доцільно обрати для пошуку злонамірної діяльності в операційних системах. Порівнюючи coach_weighted та graph_entropy очевидно буде обрати graph_entropy через більший мінімум та середнє значення інформативності.

3.5 Графічне представлення виявлених APT-атак

Приведемо результати алгоритмів кластеризації до зручного для аналізу вигляду, а саме: coach_weighted, graph_entropy, graph_entropy_weighted у графічному вигляді.

Зазначемо, якщо представлений програмний комплекс буде впроваджений в експлотацію, як описано в розділі 4, відображення злонамірної діяльності у вигляді графа значно спростить процесс детектування APT-атак, покращить функціонал програм для моніторінгу системної діяльності та спростить роботу персоналу обслуговування інформаційних систем.

Для відображення результатів використаємо наступні позначення:

- об’єкти системної або користувацької діяльності та зв’язки будуть позначатись сірим кольором з відповідною назвою;
- об’єкти злонамірної діяльності та зв’язки будуть позначатись червоним або поморанчевим кольором:
    - червоний колір відображає об’єкти які ввійшли в обраний кластер наведеного алгоритму;
    - помаранчевий колір відображає об’єкти які не війшли в обраний кластер.

Для відображення злонамірної активності використовувались кластери з найбільшою інформативнісю. Як видно з рис. 3.16 та 3.17 червоні та помаранчеві вершини зв’язані між собою та утворюють одит суцільний підграф, який займає незначну частину з усієї системної діяльності.

Оскільки рис. 3.16 та рис. 3.17 мають надлишкову інформативність виконаємо наступні перетворення даних – відфільтруємо тільки ті вершини, що пов’язані з вершинами злонамірної діяльності. Для наглядності зобразимо це на рис. 3.18 та 3.19. Візуально структура підграфів майже не змінилась, оскільки це результати для одного датасету. Відмінність тільки в кількості червоних та

помаранчевих вершин, вона зумовлена тим, що graph_entropy_weighted дав кращий результат, ніж graph_entropy.

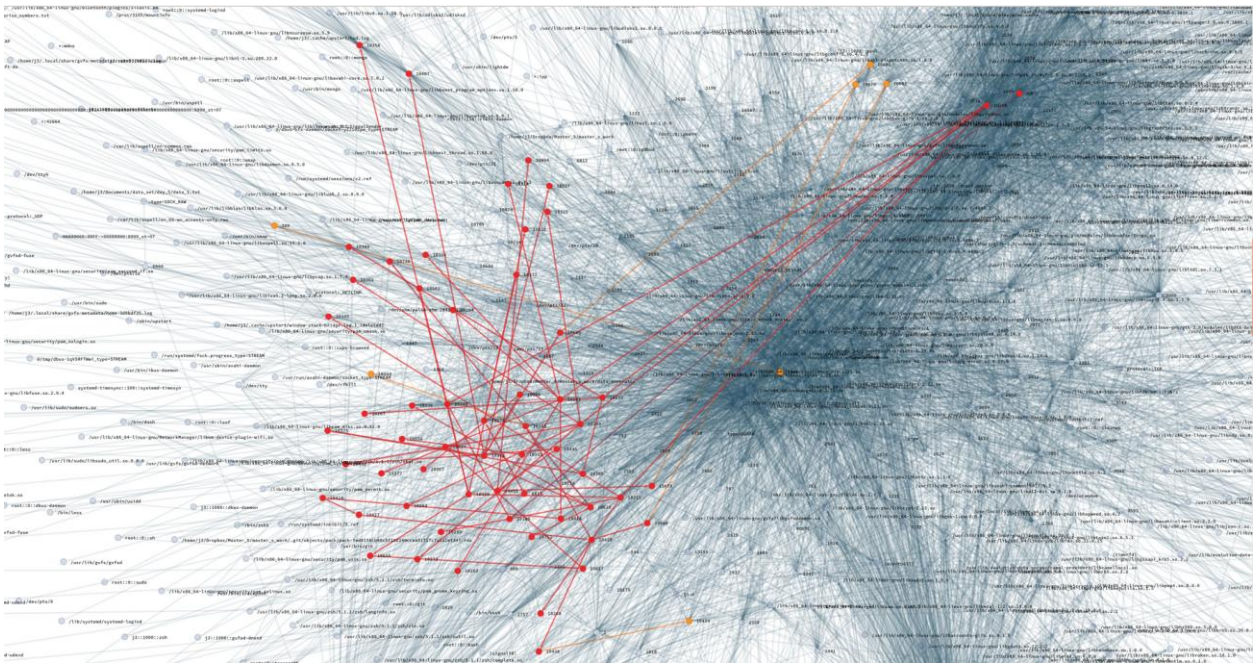

Рисунок 3.19 – Результат алгоритму graph_entropy_weighted для data_2

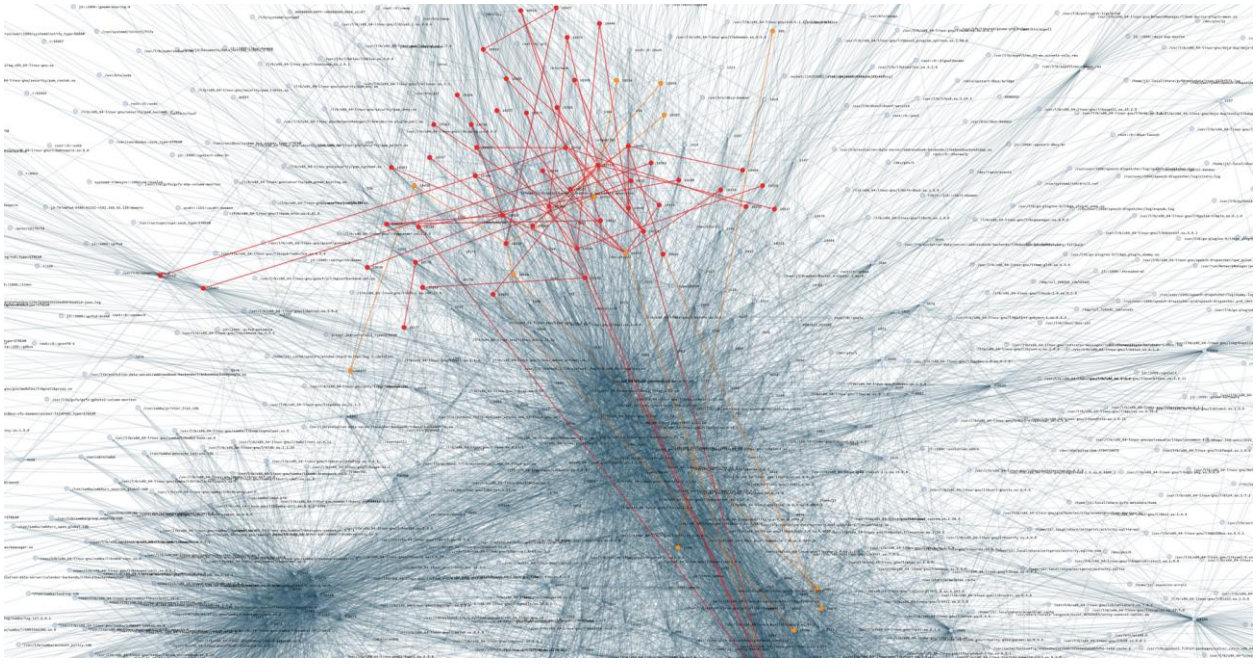

Рисунок 3.20 – Результат алгоритму graph_entropy для data_2

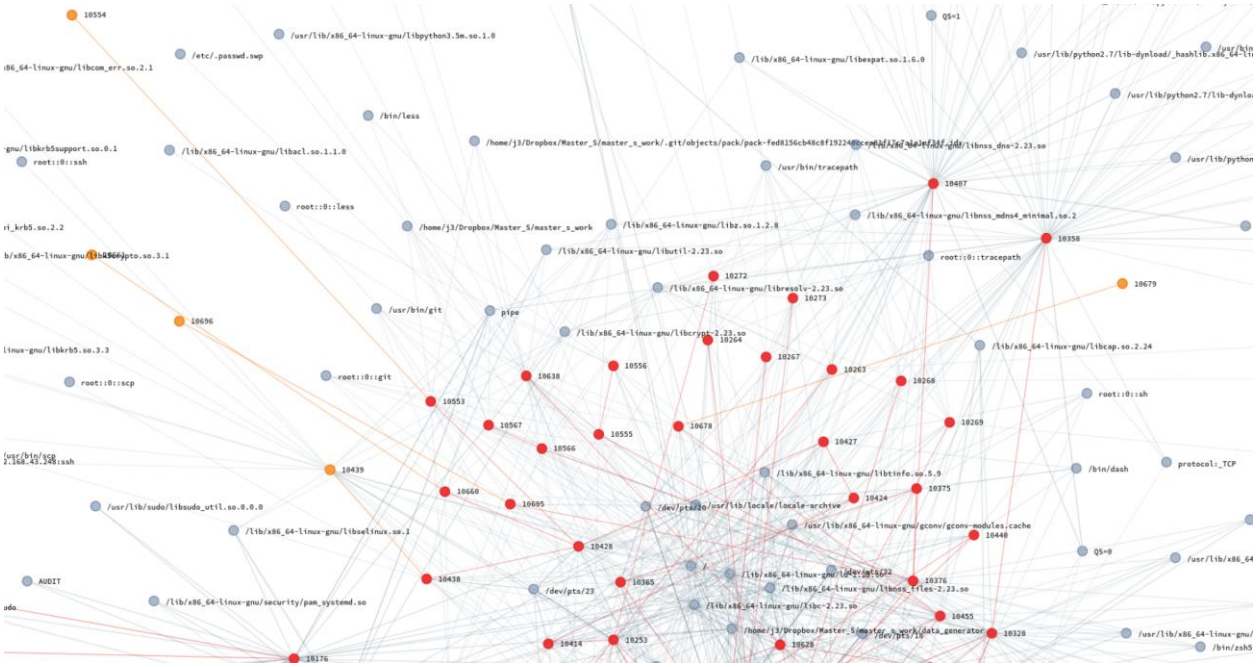

Рисунок 3.21 – Відфільтрований результат алгоритму graph_entropy_weighted для data_2

Рисунок 3.22 – Відфільтрований результат алгоритму graph_entropy для data

Наведемо візуальні приклад результатів алгоритму coach_weighted для датасетів data_1 та data_3. Для останього чітко прослідковується велика кількість

помаранчевих вершин через низький рівень інформативності обраного кластера – 45,14 %.

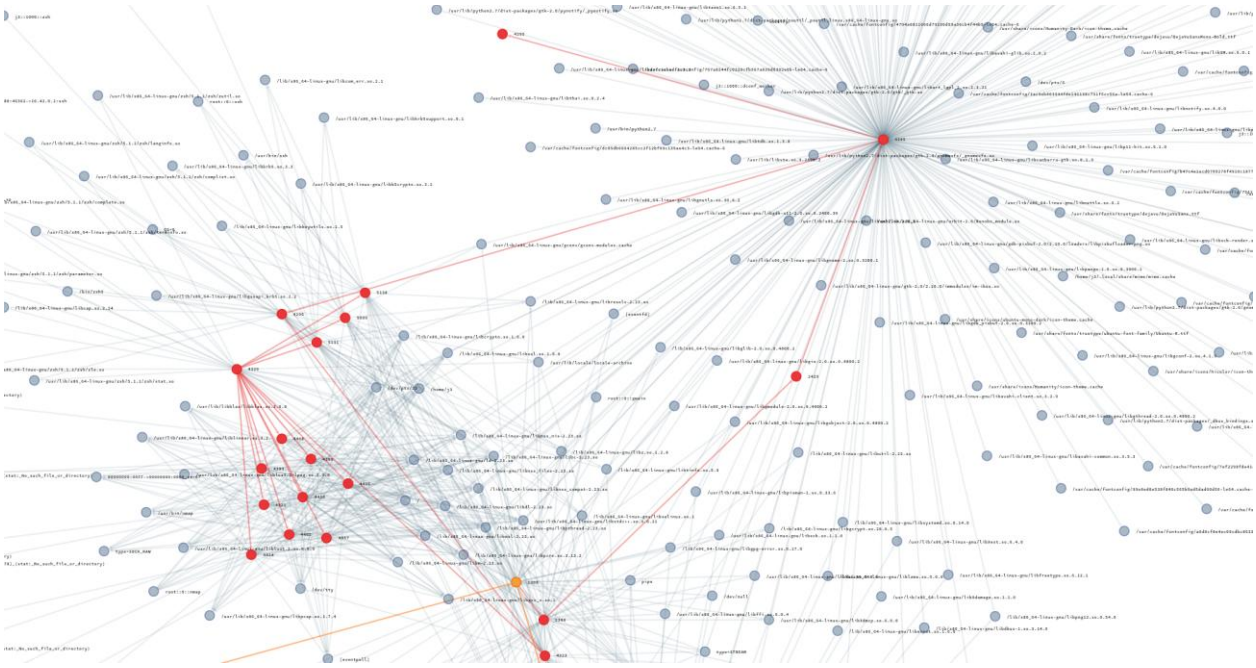

Рисунок 3.23 – Відфільтрований результат алгоритму coach_weighted для data_1

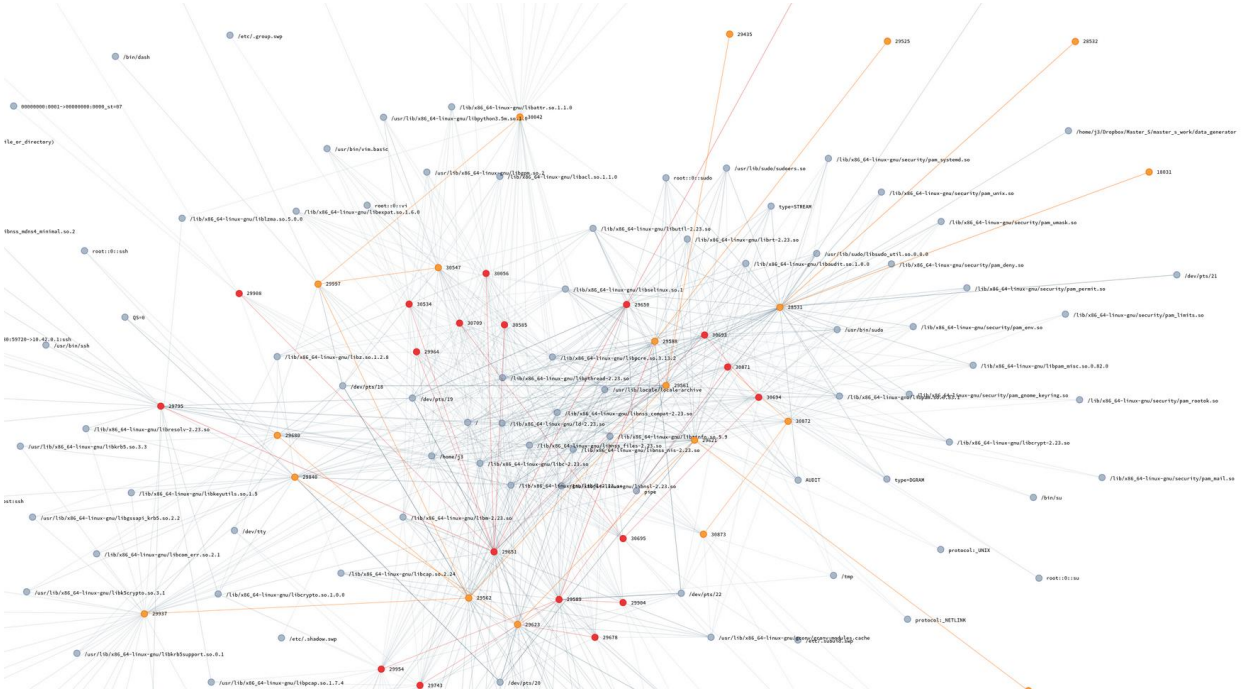

Рисунок 3.24 – Відфільтрований низький результат алгоритму coach_weighted для data_3

Висновки до розділу 3

Проведені дослідження можна використовувати в цілях безпеки операційних систем, для підвищення ефективності.

У цьому розділі було розроблено експерементальний програмний комплекс для визначення кращого алгоритму кластеризації графів для детектуваня злонамірної діяльності в операційних системах.

Для реалізації поставленої задачі:

- запропоновано працездатну схему обробки системної діяльності, та детектування злонамірної діяльності.
- запропоновано схему представлення системної діяльності у вигяді графа.
- проаналізована ефективність алгоритмів кластеризації графіф.
- запропоновано формат вихідних даних в вигляді текста для подальшого аналіза іншими програмними засобами та в вигляді графічних схем з допомогою бібліотеки graph-tool [8] для швидкого аналізу людиною.

В якості вхідних даних було зібрано п'ять різних датасетів загальним обсягом 2.4 гігабайти. Зазначемо, що в обробленому вигляді, обсяг зменшився до 3.9M. Визначена працездатність та швиткодія обраних алгоритмів.
Виходячи з інформативності результатів, coach, dpclus та dpclus_weighted не бажанно використовувати в рішеннях подібних задач.

Зроблено висновок: найкраще для поставленої задачі підходить алгоритм graph_entropy_weighted, він дав максимальний результат на кожному з обраних датасетів. Але це самий повільний з обраних алгоритмів. Для підвищення эфективності також пропонується використання алгоритма coach_weighted. В свою чергу він не надає настільки інформативний результат, як попередній алгоритм – гірше на 7% але дані оброблює набагато скоріше. Для порівняння:

- graph_entropy_weighted: 273,03s;

- coach_weighted: 0,42s.

Після побудови програмного комплексу та осмислення результатів, пропонуються наступні покращення, для прикладної реалізації:

- Винесення коду для збору даних в окремий kernel модуль для UNIX систем (уникнення використання lsof);
- Передача даних для обробки на інший пристрій через usb або ethernet з'єднання, для розвантаження основної системи;
- Переписати алгоритми кластеризації на C/C++, для підвищення швиткодії;
- Використовувати sql базу даних для підвищення швидкості обміну інформацією.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

Відповідно до попередніх розділів, мета цього розділу – опис ідеї стартап-проекту. В опис входять: аналіз ринкових можливостей, переваги та недоліки ідеї, можливість впровадження та аналіз конкурентів.

4.1 Опис ідеї проекту

APT-атаки є та будуть залишатися одними з найнебезпечніших загроз для великих IT-інфраструктур, буть то цивільний, державний або білнес сектор. Для виявлення APT-атак не так давно почали використовувати машинне навчання. За останій час було розроблено багато вдалих інструментів для детектуваня APT, більшість з них зосереджена на мережевому рівні. Ідея стартап проекту полягає в розробці інстумента виявлення APT-атак на рівні операційних систем де відбувається більша частина злонамірної діяльності з використанням машиного навчання. Пропонується побудувати програмний комплекс, який буде сумісний з усіма UNIX системами, та буде мати можливість індивідуального налаштування для покращення детектування, див. розділ 3.

Таблиця 4.1 – Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Ідея полягає у розробці інструменту, який буде ефективно, вчасно знаходити факти проведення APT-атак, тим самим впливати на безпеку інформаційних структур.	1. Застосування для звичайних користувачів 2. Застосування у інформаційних структурах бізнесу. 3. Застосування у державних інф. структурах.	• Поліпшене детектування злонамірної діяльності в операційній системі. • Детальний аналіз дій користувачів в операційній системі

Після аналізу потенійно техніко-економічних властивостей поравнянно з конкурентами в області захисту інформаційї визначено, що через новизну ідеї, конкурентів в домінуючими характеристиками не має. Наразі схожі досліди проводились виключно для системи Windows для декількох програм, а не операційної системи в цілому.

4.2 Технологічний аудит ідеї проекту

Таблиця 4.2 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1	Розробка програмного рішення виявлення APT-атак	Інвестиції інвесторів, акселераторів стартапів	Наявна	Платна, недоступна
2	Розробка програмного рішення виявлення APT-атак	Власні інвестиції	Наявна	Платна, доступна
Обрана технологія реалізації ідеї проекту: Для реалізації обрані інвестиції акселераторів стартапів, а також інвесторів, для створення групи розробки та підтримки продукту				

4.3 Аналіз ринкових можливостей для запуску стартап-проекту

Фактор загроз передбачає сукупність умов, керуючи, якими можна дійти до високої спроможності конкуренції та швидкого розвитку компанії.

Дані щодо факторів загроз наведені в таблиці 4.3.

Таблиця 4.3 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Динамічно зміні потреби користувача	Відсутність потреб в запропонованому програмному забезпеченню	Відповідна зміна програмного забезпечення відповідно до потреб користувачів
2	Висока конкуренція	Розробка п. з. з кращими показниками	Продаж компанії

Таблиця 4.4 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	Необхідність швидко та ефективно реагувати на APT-атаки. Отримувати детальну інформацію про злонамірну діяльність.	Державний сектор	Високий поріг входження	Високі вимоги до найдійності програмного забезпечення
2		Бізнес сектор	Різний підхід до роботи з програмним забезпеченням	Вимоги до підтримки та супроводу
3		Звичайні користувачі (системні адміністратори)	Мінімальний поріг входження	Вимоги до легкості впровадження та використання

Таблиця 4.5 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Підвищення покупної спроможності цільової аудиторії	Зростання зацікавленності та фінансових витрат з боку покупців	Розширення пропонуємих послуг
2	Втрата конкурентами ключових позицій	Зниження зацікавленності до продукції конкурентних компаній	Поступова монополізація ринку

Таблиця 4.6 – SWOT-аналіз стартап-проекту

Сильні сторони: Новітні підходи виявлення APT-атак з високими показниками	Слабкі сторони: З боку вітчізняних інвесторів ризиковані інвестиції
Можливості: Підвищення ефективності детектування	Загрози: Поява конкурентної продукції з удосконаленими методами виявлення APT-атак

Таблиця 4.7 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції - монополістична	Конкуренція за ринок збуту	Постійний розвиток пропозицій, боротьба за клієнтів
2. За рівнем конкурентної боротьби - міжнародний	Іноземні компанії	Вихід на іноземні ринки, залучення іноземних фахівців
3. За галузевою ознакою - внутрішньогалузева	Конкуренція ведеться виключно в даній галузі	Використовувати новітні дослідження та розробки для покращення продукту
4. Конкуренція за видами товарів: - товарно-видова	Послуги та види є однаковими в межах одного сегменту ринку	Розробити рішення та пропозицію кращу ніж в конкурентів
5. За характером конкурентних переваг - нецінова	В характеристиках продукту та його властивостях	Створити продукт, який демонструє кращі показники ніж в конкурентів
6. За інтенсивністю - марочна	На ринку присутні бренди інших компаній	Таргітована, активна реклама

4.4 Розроблення ринкової стратегії проекту

Таблиця 4.8 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	Бізнес сектор	Висока	Висока, проблема APT-атак дуже поширена серед наедених груп	Середня	Новизна та ефективність методів сприяє швидкому виходу на ринок
2	Державний сектор	Середня		Висока	
3	Сектор звичайних користувачів (системних адміністраторів)	Висока		Середня	

Таблиця 4.9 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку**
1	Надання інформаційних послуг	Стратегія охоплення: ринкове позиціонування	Точність виявлення атак, короткий час виявлення атак після проведення, детальна інформація про атаку	Стратегія диференціації

Таблиця 4.10 – Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки**
1	ні	так	ні	Зайняття конкурентної ніші

Таблиця 4.11 – Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1	Точність виявлення факту атаки	Підвищена точність, через використання нових підходів	Підвищенна точність яка базується на інформації зібраної про усі процеси в операційній системі та на ефективних методах обробки інформації
2	Короткий час виявлення атаки	Час обробки даних менший за час генерації нових даних	Виявлення атки проходить у реальному часі, в залежності від налаштувань які внес користувач, після накопичення достатньої інформації, атака може бути виявленна
3	Наявність детальної інформації про атаку	Детальна інформація на рівні ядра операційної системи	Інформація представляється у зручному текстовому або графічному вигляді та підлягає детальному аналізу за допомогою інших програм.

Таблиця 4.12 – Опис трьох рівнів моделі товару [25].

Рівні товару	*Сутність та складові*
I. Товар за задумом	Виявлення APT-атак в інформаційних системах
II. Товар у реальному виконанні	Властивості/характеристики
	1. Виявлення злонамірної діяльності на рівні операційних систем за максимально короткий час. 2. Оцінка усієї системної діяльності 3. Відображення детальних результатів в зручному форматі
	Якість: використання алгоритмів кластеризації графів, які показали найбільш ефективний результат для даної задачі
	Компанія: “Green ASM”, продукт: “APT graph detector”
III. Товар із підкріпленням	Підтримка відсутня
	Підтримка користувачів згідно ліцензії
За рахунок чого потенційний товар буде захищено від копіювання: патент на метод детектування, використання платних ліцензій на програмний продукт.	

Висновки до розділу 4

В цьому розділі магістерської роботи розглядалась реалізація стартап-проекту на основі наведених досліджень наведених в розділі 1, 2, 3 та методик описаних в документі [23].

Виходячи з наведеної інформації була наведенна ідея що до реалізації стартапу, ринкової конкурентної діяльності, таргетованої маркетингової стратегії; оцінювались ринкові можливості стартап-ідеї.

ВИСНОВКИ

В ході роботи вирішено такі задачі:

1) Розглянуто та систематизовано існуючі моделі опису APT-атак. Найбільш повноцінною з боку опису виявилась модель Cyber Kill Chain, яка враховує циклічність атак даного типу. Найбільш інформативно є модель ATT&CK Matrix разом з описом конкретного етапу, ця модель має практичні приклади та містить методології по усуненню та детектуванню атак.

В ході систематизації інформації моделей було виявлено, що переважно більша частина діяльності APT-атаки буде проводитись в середовищі операційних систем. Будь яка мережева активність з боку зловмисника буде мати повноціне відображення у атакованих операційних системах. Отже, обробляючи інформацію з оточення операційної системи, вирогідність детектування атаки значно вища.

2) Розглянуто ряд методів для детектування APT-атак на рівні операційних систем, такі як: System info monitoring, Security information and event management, File and process scanning і тд. Значним недоліком є неповне інформаційне покриття операційної системи, та пошук за шаблонами, що значно впливає на якість детектування – це підвищує вирогідність пропустити атаку з використанням zero-day вразливостей.

3) Формалізована проблема обробки інформації що до системної діяльності з різних операційних систем. Значною проблемою є недетермінованість інформації та її швидка зміна за часом, оновленями, встановленням нових програм.

4) Наведено спосіб приведення діяльності в операційних системах до детермінованих моделей. Запропоновано дві моделі:

- Відображення усіх процессів операційної системи в множену n-мірного простору, де для кожно процесу буде існувати відповідна точка з визначиними для усіх процесів параметрами.
- Відображення множени процессів в граф. Для кожного процесу або інформаційного об’єкту який використовує процес буде існувати відповідна вершина. Вага ребер відповідна до часу існування ребер.

З огляду на початкову структуру процесів для дослідження було обрано графове представлення даних.

5) Запропоновано використовувати алгоритми кластеризації графів для формування кластера, якій містить в собі максимальну кількість інформації про злонамірну діяльність, через властивості алгоритмів цього типу, а саме: здатність обробляти недетерміновану, початково невизначену інформацію, з невизначеною кількістю груп. Для перевірки працездатності запропонованого підходу було обрано наступні алгоритми: Graph Entropy, Coach, DPClu, IPCA (з використанням зваженого графа).

6) Проведено дослідження. Розроблено експерементальний програмний комплекс для емуляції дій системного адміністратора та зловмисника в рамках однієї операційної системи типу Linux. Реалізовані методи що до збору, обробки, маркування системної діяльності. Зібрано п’ять різних датасетів. Проаналізовано, результати кластеризації графів системної діяльності. Найкращі інформативні результати показали алгоритми graph_entropy_weighted та coach_weighted. Coach_weighted приблизно на 7% поступається алгоритму graph_entropy_weighted але має значну перевагу в швидкості обробки інформації.

Практичне значення результатів полягає у високій результативності використання наведених алгоритмів кластеризації графів системної діяльності для виявлення APT-атак. В результаті з’являється можливість створення та доповнення універсальних та ефективних IDS, відкаліброваних відповідно під конкретні операційні системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains [Текст] / Eric M. Hutchins , Michael J. Cloppert, Rohan M. Amin, Ph.D., — Lockheed Martin Corporation — 16 с.
2) M-Trends: the advanced persistent threat [Електронний ресурс] — Режим доступу до ресурсу: http://www.christiandve.com/wp-content/uploads/2018/05/M-Trends.pdf
3) ATT&CK Enterprise Matrix [Електронний ресурс] — Режим доступу до ресурсу: https:// https://attack.mitre.org/matrices/enterprise/
4) Finding Cyber Threats with ATT&CK-Based Analytics [Текст] / Blake E. Strom Joseph A. Battaglia Michael S. Kemmerer William Kupersanin Douglas P. Miller Craig Wampler Sean M. Whitley Ross D. Wolf — 2017 The MITRE Corporation — 53 с.
5) BZAR (Bro/Zeek ATT&CK-based Analytics and Reporting) [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/mitre-attack/car/tree/master/implementations/bzar
6) MITRE Cyber Analytics Repository [Електронний ресурс] — Режим доступу до ресурсу: https://car.mitre.org/
7) Тунелювання трафіка через DNS [Електронний ресурс] — Режим доступу до ресурсу: http://www.spy-soft.net/dns-tunneling/
8) Графічна бібліотека “graph-tool” [Електронний ресурс] — Режим доступу до ресурсу: https://graph-tool.skewed.de
9) Опис файлової системи “proc” [Електронний ресурс] — Режим доступу до ресурсу: http://man7.org/linux/man-pages/man5/proc.5.html
10) The Elements of Statistical Learning: Data Mining, Inference, and Prediction [Текст] / Editors: D. Michie, D.J. Spiegelhalter, C.C. Taylor —February 17, 1994 — 298 с.
11) A Course in Machine Learning [Текст] / Hal Daumé — Copyright 2015 — 193с.

12) Data Clustering: A Review [Текст] / A. K. Jain, M. N. Murty, P.J. Flyun — Michigan State University, Indian Institute of Sience and Ohio State University — 69 с.
13) Алгоритми кластеризації [Електронний ресурс] — Режим доступу до ресурсу: https://habr.com/post/101338/
14) Survey: Enhancing Protein Complex Prediction in PPI Networks with GO Similarity Weighting [Текст] / True Price, Francisco I Pena III, Young-Rae Cho — Department of Computer Science Baylor University Waco, TX USA — 14 с.
15) Detecting protein complexes and functional modules from protein interaction networks: A graph entropy approach [Текст] / E. C. Kenley and Y.-R. Cho — PROTEOMICS, 2011 — с. 3835–3844.
16) A coreattachment based method to detect protein complexes in PPI networks [Текст] / Wu, M., Li, X., Kwoh, C.-K. and Ng, S.-K. — BMC Bioinformatics, 2009 — 169 с.
17) Development and implementation of an algorithm for detection of protein complexes in large interaction networks [Текст] / Altaf-Ul-Amin, M., Shinbo, Y., Mihara, K., Kurokawa, K. and Kanaya, S. — BMC Bioinformatics, 2006.82 — 207 с.
18) Modifying the DPClus algorithm for identifying protein complexes based on new topological structures [Текст] / Li, M., Chen, J., Wang, J., Hu, B. and Chen, G. — BMC Bioinformatics, 2008 — 398 с.
19) Документація програми “lsof” [Електронний ресурс] — Режим доступу до ресурсу: https://linux.die.net/man/8/lsof
20) Бібліотека алгоритмів кластеризації графів [Електронний ресурс] — Режим доступу до ресурсу: https://github.com/trueprice/python-graph-clustering
21) Вразливість програми “sudo” — CVE-2019-14287 [Електронний ресурс] — Режим доступу до ресурсу: https://www.sudo.ws/alerts/minus_1_uid.html
22) Розроблення стартап-проекту: Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей [Текст] / О. А. Гавриш., 2016 - НТУУ «КПІ».

ДОДАТОК А

Реалізація тестового програмного комплексу

data_generator.py

```
#!/usr/bin/python3

import subprocess
import shlex
import sys
import time
import re
import time

#def run_command(command):
#  process = subprocess.Popen(shlex.split(command), stdout=subprocess.PIPE)
#  while True:
#    output = process.stdout.readline()
#    if output == '' and process.poll() is not None:
#      break
#    if output:
#      print(output.strip())
#  rc = process.poll()
#  return rc

def space_to__(str):
  return re.sub("\s+", "_", str.strip())

def run_command(command):
  patern = '%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s,\t%s'
  process_ID = ''         #p
  file_access_mode = ''       #a
  process_command_name = ''     #c
  file_structure_share_count = '' #C
  device_character_code = ''    #d
  device_number = ''        #D
  file_descriptor = ''      #f
  file_structure_address = ''   #F
  file_flaGs = ''         #G
  files_inode_number = ''     #i
  link_count = ''         #k
  files_lock_status = ''      #l
  process_login_name = ''     #L
  marker_b_repeated_output = ''   #m
  name_comment_addres = ''    #n
  node_identifier = ''      #N
  files_offset = ''         #o
  process_group_ID = ''       #g
  protocol_name = ''        #P
  raw_device_number = ''      #r
  parent_process_ID = ''      #R
  files_size = ''         #s
  files_stream_id = ''      #S
  files_type = ''         #t
  TCP_TPI_info = ''         #T
  process_user_ID = ''      #u
  higher_zone_name = ''       #z
  SELinux_security_contex = ''  #Z
  process = subprocess.Popen(shlex.split(command), stdout=subprocess.PIPE)
  flag_u = 0
  flag_c = 0
  flag_p = 0
  while True:
```

```
output = process.stdout
time.sleep(1)
if output:
  for l in output:
    line = str(l,'utf-8')
    k = line[0]
    v = line[1:].rstrip('\n')
    if k == 'a':
      file_access_mode = v
    if k == 'c':
      process_command_name = v
    if k == 'C':
      file_structure_share_count = v
    if k == 'd':
      device_character_code = v
    if k == 'D':
      device_number = v
    if k == 'f':
      file_descriptor = v
    if k == 'F':
      file_structure_address = v
    if k == 'G':
      file_flaGs = v
    if k == 'i':
      files_inode_number = v
    if k == 'k':
      link_count = v
    if k == 'l':
      files_lock_status = v
    if k == 'L':
      process_login_name = v
    if k == 'm':
      marker_b_repeated_output = v
    if k == 'N':
      node_identifier = v
    if k == 'o':
      files_offset = v
    if k == 'g':
      process_group_ID = v
    if k == 'P':
      protocol_name = v
    if k == 'r':
      raw_device_number = v
    if k == 'R':
      parent_process_ID = v
    if k == 's':
      files_size = v
    if k == 'S':
      files_stream_id = v
    if k == 't':
      files_type = v
    if k == 'T':
      TCP_TPI_info = v
    if k == 'u':
      process_user_ID = v
    if k == 'z':
      higher_zone_name = v
    if k == 'Z':
      SELinux_security_contex = v
    if k == 'p':
      process_ID = v
      file_access_mode = ''       #a
      process_command_name = ''     #c
      file_structure_share_count = '' #C
      device_character_code = ''    #d
      device_number = ''        #D
      file_descriptor = ''      #f
      file_structure_address = ''   #F
```

```
            file_flaGs = ''         #G
            files_inode_number = ''     #i
            link_count = ''         #k
            files_lock_status = ''      #l
            process_login_name = ''     #L
            marker_b_repeated_output = ''   #m
            name_comment_addres = ''    #n
            node_identifier = ''      #N
            files_offset = ''         #o
            process_group_ID = ''       #g
            protocol_name = ''        #P
            raw_device_number = ''      #r
            parent_process_ID = ''      #R
            files_size = ''         #s
            files_stream_id = ''      #S
            files_type = ''         #t
            TCP_TPI_info = ''         #T
            process_user_ID = ''      #u
            higher_zone_name = ''       #z
            SELinux_security_contex = ''  #Z

            flag_u = 0
            flag_c = 0
            flag_p = 0
          if k == 'n':
            name_comment_addres = v
            #if (v.find('/usr/lib', 0) == -1 and v.find('/lib', 0) == -1):
            #  name_comment_addres = '' #Ignore lib file

            #print(patern %       (process_ID,\
            #              file_access_mode,\
            #              process_command_name,\
            #                 file_structure_share_count,\
            #                device_character_code,\
            #              device_number,\
            #                file_descriptor,\
            #                file_structure_address,\
            #              file_flaGs,\
            #              files_inode_number,\
            #                link_count,\
            #                files_lock_status,\
            #              process_login_name,\
            #                 marker_b_repeated_output,\
            #              node_identifier,\
            #                files_offset,\
            #              process_group_ID,\
            #                protocol_name,\
            #                raw_device_number,\
            #              parent_process_ID,\
            #                files_size,\
            #              files_stream_id,\
            #              files_type,\
            #              TCP_TPI_info,\
            #              process_user_ID,\
            #              name_comment_addres,\
            #                higher_zone_name,\
            #                 SELinux_security_contex\
            #              ))
            #timestamp = str(int(time.time()))
            process_command_name = space_to__(process_command_name)
            name_comment_addres = space_to__(name_comment_addres)
            TCP_TPI_info = space_to__(TCP_TPI_info)
            if files_type != 'r' or files_type != 'w' or files_type != 'u':
              files_type = ''

            process_command_name = process_login_name + "::" + process_user_ID + "::" +
process_command_name
```

```
          if flag_c == 0:
            print("%s %s" % (process_command_name, process_ID));
            flag_c = 1

          if flag_p == 0:
            print("%s %s" % (parent_process_ID, process_ID));
            flag_p = 1

          if (file_access_mode != '' or file_access_mode != ' ') and name_comment_addres != '' and
files_type != '':
            print("%s     %s::%s::%s"     %     (process_ID,     files_type,     name_comment_addres,
file_access_mode));
          elif name_comment_addres != '' and files_type != '':
            print("%s %s::%s" % (process_ID, files_type, name_comment_addres));
          elif name_comment_addres != '':
            print("%s %s" % (process_ID, name_comment_addres));
          if TCP_TPI_info != '':
            print("%s %s" % (process_ID, TCP_TPI_info));
          #print("-------------------------------------------------")
  Return

run_command("lsof -FacCdDfFGiklLmnNopgPrRsStTuzZ -r1")
```

analyze.py

```
#!/usr/bin/python3

import sys

cluster_find_barrier = 0
def analize_res():
  data_file = sys.argv[1]
  data_ps_list = sys.argv[2]
  num_lines = sum(1 for line in open(data_file))
  d_list = data_ps_list.split(" ")


  d_list_size = len(d_list)
  c_list_size = 0
  c_pid_list_size = 0
  d_count = 0
  i = 0

  print("Total,Uniformity,optimality")
  try:
    with open(data_file, 'r') as fp:
      line = fp.readline()
      cnt = 1
      while line:
        i = i + 1
        d_count = 0
        c_pid_list_size = 0
        list = line.split(" ")
        c_list_size = len(list)
        for w in list:
          if w.isdigit():
            c_pid_list_size = c_pid_list_size + 1
            if w in d_list:
              d_count = d_count + 1
        total_find = (d_count / d_list_size) * 100
        if c_pid_list_size == 0:
          cluster_find = 0
        else:
          cluster_find = (d_count / c_pid_list_size) * 100

        if total_find > 0 and cluster_find > cluster_find_barrier:
          print(str(round(total_find,   2))   +   ","   +   str(round(cluster_find,   2))   +   ","   +
str(round(total_find * cluster_find / 100, 2)))

        line = fp.readline()
  finally:
    fp.close()
  return
```

```
def main():
  analize_res()

main()
```

view_graph.py

```
#!/usr/bin/python3

import graph_tool.all as gt
import pprint
import datetime

graph_list=[]
vertex_list=[]
#Data should be sorted
def collect_data(node_1, node_2, node_1_color, node_2_color, weight):
  #print(node_1, node_2, node_1_color, node_2_color, weight)
  post = {'node_1': node_1,
   'node_1_color': node_1_color,
   'node_2': node_2,
   'node_2_color': node_2_color,
   'weight': weight}
  graph_list.insert(0, post)

def create_vertex_list():
  i = len(graph_list)
  k = 0
  for g in graph_list:
    if g['node_1'] not in vertex_list:
      vertex_list.insert(0, g['node_1'])
    if g['node_2'] not in vertex_list:
      vertex_list.insert(0, g['node_2'])
    print(str(i) + "/" + str(k))
    k = k + 1
  print("Done: create_vertex_list")
  print(vertex_list)
  print(len(vertex_list))

#node_1 node_2 node_1_color node_2_color weight
#"root::0::winbindd 1880 green gray 100"
def get_data_from_file():
  data_file = input("Print full file path : ")
  num_lines = sum(1 for line in open(data_file))
  i = 0
  j = 0
  try:
    with open(data_file, 'r') as fp:
      line = fp.readline()
      cnt = 1
      while line:
        i = i + 1
        j = j + 1
        list = line.split(" ")
        list[4] = list[4].replace('\n', '')
        if list[0] != '' and list[1] != '' and list[2] != '' and list[3] != '' and list[4] != '':
          collect_data(list[0], list[1], list[2], list[3], list[4])
        line = fp.readline()
        if i == 100:
          i = 0
          print(str(num_lines) + "/" + str(j))
      print("Done: collect_data")

  finally:
    fp.close()
  return

def create_graph():
  g = gt.Graph(directed=True)

  v_prop = g.new_vertex_property("string")
  e_len = g.new_edge_property("double")
  partition = g.new_vertex_property('int')
  vertex_color = g.new_vertex_property('vector<double>')
  g.vertex_properties['vertex_color'] = vertex_color
  vertex_fill_color = g.new_vertex_property('vector<double>')
```

```
  g.vertex_properties['vertex_fill_color'] = vertex_fill_color
  edge_color = g.new_edge_property('vector<double>')
  g.edge_properties['vertex_color'] = edge_color


  weight = 0
  for data in graph_list:
    if weight < int(data['weight']):
      weight = int(data['weight'])

  for v in vertex_list:
    vert = g.add_vertex()
    v_prop[vert] = v
    vertex_color[vert] = (0x27/0xFF,0x3A/0xFF,0x9E/0xFF,0.3) #273A9E blue
    vertex_fill_color[vert] = (0x2E/0xFF,0x55/0xFF,0x6A/0xFF,0.2) #2E556A gray

  j = len(vertex_list)
  k = 0
  ogange_edge = 0
  for v1 in g.vertices():
    name_1 = v_prop[v1]
    for data in graph_list:
      if name_1 == data['node_1']:
        name_2 = data['node_2']
        for v2 in g.vertices():
          if v_prop[v2] == name_2:
            e = g.add_edge(v1, v2)

            if data['node_1_color'] == 'orange' and data['node_1'].isdigit():
              vertex_color[v1] = (0xFF/0xFF,0x6B/0xFF,0x00/0xFF,0.80) #FF6B00 ogange
              vertex_fill_color[v1] = (0xF3/0xFF,0x96/0xFF,0x29/0xFF,0.95) #F39629 ogange

            if data['node_2_color'] == 'orange' and data['node_2'].isdigit():
              vertex_color[v2] = (0xFF/0xFF,0x6B/0xFF,0x00/0xFF,0.80) #FF6B00 ogange
              vertex_fill_color[v2] = (0xF3/0xFF,0x96/0xFF,0x29/0xFF,0.95) ##F39629 ogange

            if data['node_1_color'] == 'red' and data['node_1'].isdigit():
              vertex_color[v1] = (0xe8/0xFF,0x22/0xFF,0x25/0xFF,0.80) #e82225red
              vertex_fill_color[v1] = (0xe8/0xFF,0x22/0xFF,0x25/0xFF,0.95) #e82225red

            if data['node_2_color'] == 'red' and data['node_2'].isdigit():
              vertex_color[v2] = (0xe8/0xFF,0x22/0xFF,0x25/0xFF,0.80) #e82225red
              vertex_fill_color[v2] = (0xe8/0xFF,0x22/0xFF,0x25/0xFF,0.95) #e82225red

            if data['node_1_color'] == 'orange' and data['node_2_color'] == 'orange' and
data['node_2'].isdigit() and data['node_1'].isdigit():
              w = (int(data['weight'])/(weight * 3)) + 0.5
              edge_color[e] = (0xFF/0xFF,0x6B/0xFF,0x00/0xFF,w) #FF6B00 ogange
              ogange_edge = ogange_edge + 1
            elif data['node_1_color'] == 'red' and data['node_2_color'] == 'red' and
data['node_2'].isdigit() and data['node_1'].isdigit():
              w = (int(data['weight'])/(weight * 3)) + 0.66
              edge_color[e] = (0xe8/0xFF,0x22/0xFF,0x25/0xFF,w) #e82225red
            else:
              e_len[e] = int(data['weight'])
              w = (int(data['weight'])/(weight * 2)) + 0.1
              edge_color[e] = (0x2E/0xFF,0x55/0xFF,0x6A/0xFF,w) #273A9E gray

            break;

    print(str(j) + "/" + str(k))
    k = k + 1
  print("ogange_edge: " + str(ogange_edge))

  #pos = gt.sfdp_layout(g)
  gt.graph_draw(g, #pos=pos,
         bg_color=[1,1,1,1],
         vertex_text_position=0,
         vertex_size=22,
         vertex_text=v_prop,
         vertex_font_size=16,
         vertex_fill_color=vertex_fill_color,#(0x2E/0xFF,0x55/0xFF,0x6A/0xFF,0.7), #2E556A gray
         vertex_color=g.vertex_properties['vertex_color'],
         edge_color=edge_color,
         edge_pen_width=5,
         #bg_color=[0,0,0,1],
         output_size=(16000, 16000),
         output="graph_pid.png")
  print("Max weight:" + str(weight))
```

```
  print("Done: create_graph")

get_data_from_file()
create_vertex_list()
create_graph()
```

mark_graph.py

```
#!/usr/bin/python3

from pymongo import MongoClient
import pprint
import datetime
client = MongoClient('mongodb://localhost:27017/')
db = client['graph_data_set']
collection_name = input("Print collection name : ")
collection = db[collection_name]
#{ "_id" : ObjectId("5da8d0400cb47cea0ad06201"), "node_1" : "root::0::winbindd", "node_2" : "1937",
"node_2_color" : "green", "time_end" : "23", "node_1_color" : "green", "time_start" : "23", "weight"
: 1 }
def mark():
  color = input("Enter color[red orange yellow green]: ")
  nodes = input("Enter node with next format 'node_1 node_2 node_3'... : ")
  list = nodes.split(" ")
  for l in list:
    if l != '' and l != ' ' and l != '  ':
      myquery = { "node_1": l }
      newvalues = { "$set": { "node_1_color": color,  "node_2_color": color} }
      collection.update_many(myquery, newvalues)

      myquery = { "node_2": l }
      newvalues = { "$set": { "node_1_color": color,  "node_2_color": color} }
      collection.update_many(myquery, newvalues)
  print("Done: mark")

def main():
  mark()
  return 0


main()
```

get_db.py

```
#!/usr/bin/python3

from pymongo import MongoClient
import pprint
import datetime
client = MongoClient('mongodb://localhost:27017/')
db = client['graph_data_set']
collection_name = input("Print collection name : ")
collection = db[collection_name]
##  node_1 node_2 node_1_color node_2_color weight
def get_data_from_db():
  file = input("Print full file name : ")
  f = open(file, "w")
  for post in collection.find():
    #f.write(post['node_1']  +  "  "  +  post['node_2']  +  "  "  +  post['node_1_color']  +  "  "  +
post['node_2_color'] + " " + str(post['weight']) + "\n")
    f.write(post['node_1']  +  "  "  +  post['node_2']  +  "  "  +  post['node_1_color']  +  "  "  +
post['node_2_color'] + " " + str(post['time_start']) + "\n")
  f.close()
  return 0

def main():
  return get_data_from_db()

main()
```

set_db.py

```
#!/usr/bin/python3
from pymongo import MongoClient
```

```
import pprint
import datetime

client = MongoClient('mongodb://localhost:27017/')
db = client['graph_data_set']
collection_name = input("Print collection name : ")
collection = db[collection_name]
db_list = []
def get_list_index(node_1, node_2):
  res_list=[i for i in range(len(db_list)) if db_list[i]['node_1'] == node_1 and
db_list[i]['node_2'] == node_2]
  return res_list

def collect_data(node_1, node_2, time):
  weight = 0

  res_list = get_list_index(node_1, node_2)
  if len(res_list) == 0:
    post = {'node_1': node_1,
     'node_1_color': 'green',
     'node_2': node_2,
     'node_2_color': 'green',
     'weight': 1,
     'time_start': time,
     'time_end': time}
    db_list.insert(0, post)
  elif len(res_list) == 1:
    index = res_list[0]
    db_list[index]['time_end'] = time
    weight = db_list[index]['weight'] + 1
    db_list[index]['weight'] = weight
  else:
    print("Error index > 1")
    exit(1)

def set_data_to_db():
  x = collection.insert_many(db_list)
  print("Done: set_data_to_db")

def get_data_from_file():
  data_file = input("Print full file path : ")
  num_lines = sum(1 for line in open(data_file))
  i = 0
  j = 0
  try:
    with open(data_file, 'r') as fp:
      line = fp.readline()
      cnt = 1
      while line:
        i = i + 1
        j = j + 1
        list = line.split(" ")
        list[2] = list[2].replace('\n', '')
        if list[0] != '' and list[1] != '' and list[2] != '':
          collect_data(list[0], list[1], list[2])
        line = fp.readline()
        if i == 100:
          i = 0
          print(str(num_lines) + "/" + str(j))
      print("Done: collect_data")
      set_data_to_db()
      print("Done: set_data_to_db")
  finally:
    fp.close()
  return

def main():
  t_start = datetime.datetime.now().time()
  get_data_from_file()
  print("Start:")
  print(t_start)
  print("End:")
  print(datetime.datetime.now().time())
  return 0

main()
```

ДОДАТОК Б

Оброблена та відсортована системна діяльність

```
9578 [eventfd] 1940
9578 /home/j3 970
9578 /lib/x86_64-linux-gnu/ld-2.23.so 970
9578 /lib/x86_64-linux-gnu/libc-2.23.so 970
9578 /lib/x86_64-linux-gnu/libdbus-1.so.3.14.6 970
9578 /lib/x86_64-linux-gnu/libdl-2.23.so 970
9578 /lib/x86_64-linux-gnu/libgcc_s.so.1 970
9578 /lib/x86_64-linux-gnu/libgcrypt.so.20.0.5 970
9578 /lib/x86_64-linux-gnu/libglib-2.0.so.0.4800.2 970
9578 /lib/x86_64-linux-gnu/libgpg-error.so.0.17.0 970
9578 /lib/x86_64-linux-gnu/libjson-c.so.2.0.0 970
9578 /lib/x86_64-linux-gnu/liblzma.so.5.0.0 970
9578 /lib/x86_64-linux-gnu/libm-2.23.so 970
9578 /lib/x86_64-linux-gnu/libnsl-2.23.so 970
9578 /lib/x86_64-linux-gnu/libpcre.so.3.13.2 970
9578 /lib/x86_64-linux-gnu/libpthread-2.23.so 970
9578 /lib/x86_64-linux-gnu/libresolv-2.23.so 970
9578 /lib/x86_64-linux-gnu/librt-2.23.so 970
9578 /lib/x86_64-linux-gnu/libselinux.so.1 970
9578 /lib/x86_64-linux-gnu/libsystemd.so.0.14.0 970
9578 /lib/x86_64-linux-gnu/libwrap.so.0.7.6 970
9578 pipe 5820
9578 /run/user/1000/speech-dispatcher/log/espeak.log 1940
9578 /run/user/1000/speech-dispatcher/log/speech-dispatcher.log 970
9578 /run/user/1000/speech-dispatcher/pid/speech-dispatcher.pid_(deleted) 970
9578 type=STREAM 970
9578 /usr/lib/locale/C.UTF-8/LC_CTYPE 970
9578 /usr/lib/locale/locale-archive 970
9578 /usr/lib/speech-dispatcher-modules/sd_espeak 970
9578 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache 970
9578 /usr/lib/x86_64-linux-gnu/libasound.so.2.0.0 970
9578 /usr/lib/x86_64-linux-gnu/libasyncns.so.0.3.1 970
9578 /usr/lib/x86_64-linux-gnu/libdotconf.so.0.0.1 970
9578 /usr/lib/x86_64-linux-gnu/libespeak.so.1.1.48 970
9578 /usr/lib/x86_64-linux-gnu/libFLAC.so.8.3.0 970
9578 /usr/lib/x86_64-linux-gnu/libgthread-2.0.so.0.4800.2 970
9578 /usr/lib/x86_64-linux-gnu/libjack.so.0.1.0 970
9578 /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1 970
9578 /usr/lib/x86_64-linux-gnu/libogg.so.0.8.2 970
9578 /usr/lib/x86_64-linux-gnu/libportaudio.so.2.0.0 970
9578 /usr/lib/x86_64-linux-gnu/libpulse-simple.so.0.1.0 970
9578 /usr/lib/x86_64-linux-gnu/libpulse.so.0.19.0 970
9578 /usr/lib/x86_64-linux-gnu/libsndfile.so.1.0.25 970
9578 /usr/lib/x86_64-linux-gnu/libsonic.so.0.2.0 970
9578 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21 970
9578 /usr/lib/x86_64-linux-gnu/libvorbisenc.so.2.0.11 970
9578 /usr/lib/x86_64-linux-gnu/libvorbis.so.0.4.8 970
9578 /usr/lib/x86_64-linux-gnu/libXau.so.6.0.0 970
9578 /usr/lib/x86_64-linux-gnu/libxcb.so.1.1.0 970
9578 /usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0 970
9578 /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-8.0.so 970
9578 /usr/lib/x86_64-linux-gnu/speech-dispatcher/spd_pulse.so 970
9583 / 582
9583 /dev/shm/pulse-shm-2076972276 582
9583 /dev/shm/pulse-shm-3794718478 582
9583 [eventfd] 1164
9583 /home/j3 582
9583 /lib/x86_64-linux-gnu/ld-2.23.so 582
9583 /lib/x86_64-linux-gnu/libc-2.23.so 582
9583 /lib/x86_64-linux-gnu/libdbus-1.so.3.14.6 582
9583 /lib/x86_64-linux-gnu/libdl-2.23.so 582
9583 /lib/x86_64-linux-gnu/libgcrypt.so.20.0.5 582
9583 /lib/x86_64-linux-gnu/libglib-2.0.so.0.4800.2 582
9583 /lib/x86_64-linux-gnu/libgpg-error.so.0.17.0 582
9583 /lib/x86_64-linux-gnu/libjson-c.so.2.0.0 582
9583 /lib/x86_64-linux-gnu/liblzma.so.5.0.0 582
9583 /lib/x86_64-linux-gnu/libm-2.23.so 582
9583 /lib/x86_64-linux-gnu/libnsl-2.23.so 582
9583 /lib/x86_64-linux-gnu/libpcre.so.3.13.2 582
```

```
9583 /lib/x86_64-linux-gnu/libpthread-2.23.so 582
9583 /lib/x86_64-linux-gnu/libresolv-2.23.so 582
9583 /lib/x86_64-linux-gnu/librt-2.23.so 582
9583 /lib/x86_64-linux-gnu/libselinux.so.1 582
9583 /lib/x86_64-linux-gnu/libsystemd.so.0.14.0 582
9583 /lib/x86_64-linux-gnu/libwrap.so.0.7.6 582
9583 pipe 4656
9583 /run/user/1000/speech-dispatcher/log/dummy.log 1164
9583 /run/user/1000/speech-dispatcher/log/espeak.log 582
9583 /run/user/1000/speech-dispatcher/log/speech-dispatcher.log 582
9583 /run/user/1000/speech-dispatcher/pid/speech-dispatcher.pid_(deleted) 582
9583 type=STREAM 582
9583 /usr/lib/speech-dispatcher-modules/sd_dummy 582
9583 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache 582
9583 /usr/lib/x86_64-linux-gnu/libasyncns.so.0.3.1 582
9583 /usr/lib/x86_64-linux-gnu/libdotconf.so.0.0.1 582
9583 /usr/lib/x86_64-linux-gnu/libFLAC.so.8.3.0 582
9583 /usr/lib/x86_64-linux-gnu/libgthread-2.0.so.0.4800.2 582
9583 /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1 582
9583 /usr/lib/x86_64-linux-gnu/libogg.so.0.8.2 582
9583 /usr/lib/x86_64-linux-gnu/libpulse-simple.so.0.1.0 582
9583 /usr/lib/x86_64-linux-gnu/libpulse.so.0.19.0 582
9583 /usr/lib/x86_64-linux-gnu/libsndfile.so.1.0.25 582
9583 /usr/lib/x86_64-linux-gnu/libvorbisenc.so.2.0.11 582
9583 /usr/lib/x86_64-linux-gnu/libvorbis.so.0.4.8 582
9583 /usr/lib/x86_64-linux-gnu/libXau.so.6.0.0 582
9583 /usr/lib/x86_64-linux-gnu/libxcb.so.1.1.0 582
9583 /usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0 582
9583 /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-8.0.so 582
9583 /usr/lib/x86_64-linux-gnu/speech-dispatcher/spd_pulse.so 582
9586 / 582
9586 /dev/shm/pulse-shm-3225478376 582
9586 /dev/shm/pulse-shm-3794718478 582
9586 [eventfd] 1164
9586 /home/j3 582
9586 /lib/x86_64-linux-gnu/ld-2.23.so 582
9586 /lib/x86_64-linux-gnu/libc-2.23.so 582
9586 /lib/x86_64-linux-gnu/libdbus-1.so.3.14.6 582
9586 /lib/x86_64-linux-gnu/libdl-2.23.so 582
9586 /lib/x86_64-linux-gnu/libgcrypt.so.20.0.5 582
9586 /lib/x86_64-linux-gnu/libglib-2.0.so.0.4800.2 582
9586 /lib/x86_64-linux-gnu/libgpg-error.so.0.17.0 582
9586 /lib/x86_64-linux-gnu/libjson-c.so.2.0.0 582
9586 /lib/x86_64-linux-gnu/liblzma.so.5.0.0 582
9586 /lib/x86_64-linux-gnu/libm-2.23.so 582
9586 /lib/x86_64-linux-gnu/libnsl-2.23.so 582
9586 /lib/x86_64-linux-gnu/libpcre.so.3.13.2 582
9586 /lib/x86_64-linux-gnu/libpthread-2.23.so 582
9586 /lib/x86_64-linux-gnu/libresolv-2.23.so 582
9586 /lib/x86_64-linux-gnu/librt-2.23.so 582
9586 /lib/x86_64-linux-gnu/libselinux.so.1 582
9586 /lib/x86_64-linux-gnu/libsystemd.so.0.14.0 582
9586 /lib/x86_64-linux-gnu/libwrap.so.0.7.6 582
9586 pipe 6984
9586 /run/user/1000/speech-dispatcher/log/cicero.log 1164
9586 /run/user/1000/speech-dispatcher/log/dummy.log 582
9586 /run/user/1000/speech-dispatcher/log/espeak.log 582
9586 /run/user/1000/speech-dispatcher/log/speech-dispatcher.log 582
9586 /run/user/1000/speech-dispatcher/pid/speech-dispatcher.pid_(deleted) 582
9586 type=STREAM 582
9586 /usr/lib/speech-dispatcher-modules/sd_cicero 582
9586 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache 582
9586 /usr/lib/x86_64-linux-gnu/libasyncns.so.0.3.1 582
9586 /usr/lib/x86_64-linux-gnu/libdotconf.so.0.0.1 582
9586 /usr/lib/x86_64-linux-gnu/libFLAC.so.8.3.0 582
9586 /usr/lib/x86_64-linux-gnu/libgthread-2.0.so.0.4800.2 582
9586 /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1 582
9586 /usr/lib/x86_64-linux-gnu/libogg.so.0.8.2 582
9586 /usr/lib/x86_64-linux-gnu/libpulse-simple.so.0.1.0 582
9586 /usr/lib/x86_64-linux-gnu/libpulse.so.0.19.0 582
9586 /usr/lib/x86_64-linux-gnu/libsndfile.so.1.0.25 582
9586 /usr/lib/x86_64-linux-gnu/libvorbisenc.so.2.0.11 582
9586 /usr/lib/x86_64-linux-gnu/libvorbis.so.0.4.8 582
9586 /usr/lib/x86_64-linux-gnu/libXau.so.6.0.0 582
9586 /usr/lib/x86_64-linux-gnu/libxcb.so.1.1.0 582
9586 /usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0 582
9586 /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-8.0.so 582
9586 /usr/lib/x86_64-linux-gnu/speech-dispatcher/spd_pulse.so 582
9590 / 582
```

```
9590 /dev/shm/pulse-shm-1322794211 582
9590 /dev/shm/pulse-shm-3794718478 582
9590 [eventfd] 1164
9590 /home/j3 582
9590 /lib/x86_64-linux-gnu/ld-2.23.so 582
9590 /lib/x86_64-linux-gnu/libc-2.23.so 582
9590 /lib/x86_64-linux-gnu/libdbus-1.so.3.14.6 582
9590 /lib/x86_64-linux-gnu/libdl-2.23.so 582
9590 /lib/x86_64-linux-gnu/libgcrypt.so.20.0.5 582
9590 /lib/x86_64-linux-gnu/libglib-2.0.so.0.4800.2 582
9590 /lib/x86_64-linux-gnu/libgpg-error.so.0.17.0 582
9590 /lib/x86_64-linux-gnu/libjson-c.so.2.0.0 582
9590 /lib/x86_64-linux-gnu/liblzma.so.5.0.0 582
9590 /lib/x86_64-linux-gnu/libm-2.23.so 582
9590 /lib/x86_64-linux-gnu/libnsl-2.23.so 582
9590 /lib/x86_64-linux-gnu/libpcre.so.3.13.2 582
9590 /lib/x86_64-linux-gnu/libpthread-2.23.so 582
9590 /lib/x86_64-linux-gnu/libresolv-2.23.so 582
9590 /lib/x86_64-linux-gnu/librt-2.23.so 582
9590 /lib/x86_64-linux-gnu/libselinux.so.1 582
9590 /lib/x86_64-linux-gnu/libsystemd.so.0.14.0 582
9590 /lib/x86_64-linux-gnu/libwrap.so.0.7.6 582
9590 pipe 6984
9590 /run/user/1000/speech-dispatcher/log/cicero.log 582
9590 /run/user/1000/speech-dispatcher/log/dummy.log 582
9590 /run/user/1000/speech-dispatcher/log/espeak.log 582
9590 /run/user/1000/speech-dispatcher/log/generic.log 1164
9590 /run/user/1000/speech-dispatcher/log/speech-dispatcher.log 582
9590 /run/user/1000/speech-dispatcher/pid/speech-dispatcher.pid_(deleted) 582
9590 type=STREAM 582
9590 /usr/lib/speech-dispatcher-modules/sd_generic 582
9590 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache 582
9590 /usr/lib/x86_64-linux-gnu/libasyncns.so.0.3.1 582
9590 /usr/lib/x86_64-linux-gnu/libdotconf.so.0.0.1 582
9590 /usr/lib/x86_64-linux-gnu/libFLAC.so.8.3.0 582
9590 /usr/lib/x86_64-linux-gnu/libgthread-2.0.so.0.4800.2 582
9590 /usr/lib/x86_64-linux-gnu/libltdl.so.7.3.1 582
9590 /usr/lib/x86_64-linux-gnu/libogg.so.0.8.2 582
9590 /usr/lib/x86_64-linux-gnu/libpulse-simple.so.0.1.0 582
9590 /usr/lib/x86_64-linux-gnu/libpulse.so.0.19.0 582
9590 /usr/lib/x86_64-linux-gnu/libsndfile.so.1.0.25 582
9590 /usr/lib/x86_64-linux-gnu/libvorbisenc.so.2.0.11 582
9590 /usr/lib/x86_64-linux-gnu/libvorbis.so.0.4.8 582
9590 /usr/lib/x86_64-linux-gnu/libXau.so.6.0.0 582
9590 /usr/lib/x86_64-linux-gnu/libxcb.so.1.1.0 582
9590 /usr/lib/x86_64-linux-gnu/libXdmcp.so.6.0.0 582
9590 /usr/lib/x86_64-linux-gnu/pulseaudio/libpulsecommon-8.0.so 582
9590 /usr/lib/x86_64-linux-gnu/speech-dispatcher/spd_pulse.so 582
9593 / 776
9593 /dev/null 1164
9593 /lib/x86_64-linux-gnu/ld-2.23.so 388
9593 /lib/x86_64-linux-gnu/libc-2.23.so 388
9593 /lib/x86_64-linux-gnu/libdl-2.23.so 388
9593 /lib/x86_64-linux-gnu/libglib-2.0.so.0.4800.2 388
9593 /lib/x86_64-linux-gnu/libpcre.so.3.13.2 388
9593 /lib/x86_64-linux-gnu/libpthread-2.23.so 388
9593 pipe 3880
9593 /run/user/1000/speech-dispatcher/log/cicero.log 388
9593 /run/user/1000/speech-dispatcher/log/dummy.log 388
9593 /run/user/1000/speech-dispatcher/log/espeak.log 388
9593 /run/user/1000/speech-dispatcher/log/generic.log 388
9593 /run/user/1000/speech-dispatcher/log/speech-dispatcher.log 776
9593 /run/user/1000/speech-dispatcher/pid/speech-dispatcher.pid 388
9593 /run/user/1000/speech-dispatcher/pid/speech-dispatcher.pid_(deleted) 388
9593 /run/user/1000/speech-dispatcher/speechd.sock_type=STREAM 388
9593 /usr/bin/speech-dispatcher 388
9593 /usr/lib/locale/locale-archive 388
9593 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache 388
9593 /usr/lib/x86_64-linux-gnu/libdotconf.so.0.0.1 388
9593 /usr/lib/x86_64-linux-gnu/libgmodule-2.0.so.0.4800.2 388
9593 /usr/lib/x86_64-linux-gnu/libgthread-2.0.so.0.4800.2 388
9597 / 96
9597 /proc/9597/exe 48
9639 / 388
9639 /proc/9639/exe 194
9640 / 388
9640 /proc/9640/exe 194
9654 / 344
9654 /proc/9654/exe 172
```

```
9655 / 388
9655 /proc/9655/exe 194
9656 / 388
9656 /proc/9656/exe 194
9657 / 214
9657 /proc/9657/exe 107
9658 / 122
9658 /proc/9658/exe 61
9661 / 234
9661 /proc/9661/exe 117
9763 / 388
9763 /proc/9763/exe 194
9768 / 388
9768 /proc/9768/exe 194
9910 / 62
9910 /proc/9910/exe 31
9980 / 332
9980 /proc/9980/exe 166
9981 / 388
9981 /proc/9981/exe 194
::999::Backgro.kSource 4179 195
::999::clientcursormon 4179 195
::999::Collect.xecutor 4179 195
::999::conn66 4179 19
::999::conn67 4179 27
::999::conn68 4179 19
::999::conn69 4179 11
::999::conn70 4179 8
::999::DeadlineMonitor 4179 195
::999::FlowCon.fresher 4179 195
::999::FreeMonHTTP-0 4179 195
::999::FreeMonNet 4179 195
::999::FreeMon.ocessor 4179 195
::999::ftdc 4179 195
::999::listener 4179 195
::999::Logical.cheReap 4179 195
::999::Logical.Refresh 4179 195
::999::mongod 4179 1950
::999::Periodi.kRunner 4179 195
::999::signalP.gThread 4179 195
::999::startPe.actions 4179 195
::999::startPe.ressure 4179 195
::999::Timesta.Monitor 4179 195
::999::TTLMonitor 4179 195
::999::waitForMajority 4179 195
::999::WTCheck.tThread 4179 195
::999::WTIdleS.Sweeper 4179 195
::999::WTJourn.Flusher 4179 195
9 /proc/9/exe 195
avahi::111::avahi-daemon 1137 195
avahi::111::avahi-daemon 1168 195
colord::113::colord 1686 195
colord::113::gdbus 1686 195
colord::113::gmain 1686 195
j3::1000::alsa-sink-CX207 2966 195
j3::1000::alsa-source-CX2 2966 195
j3::1000::at-spi2-registr 2837 195
j3::1000::at-spi-bus-laun 2814 195
j3::1000::bamfdaemon 2634 195
j3::1000::compiz 2863 195
j3::1000::dbus-daemon 2551 195
j3::1000::dbus-daemon 2824 195
j3::1000::dconf-service 2856 195
j3::1000::dconf_worker 13171 194
j3::1000::dconf_worker 2634 195
j3::1000::dconf_worker 3354 195
j3::1000::dconf_worker 3373 195
j3::1000::dconf_worker 3511 195
j3::1000::dconf_worker 4463 195
j3::1000::dconf_worker 4474 195
j3::1000::dconf_worker 4476 195
j3::1000::dconf_worker 4554 195
j3::1000::dconf_worker 6802 195
j3::1000::dconf_worker 8332 195
j3::1000::deja-dup-monito 4554 195
j3::1000::evolution-addre 3001 390
j3::1000::evolution-addre 3012 390
j3::1000::evolution-calen 2955 390
j3::1000::evolution-calen 2962 585
```

```
j3::1000::evolution-calen 3004 390
j3::1000::evolution-sourc 2932 195
j3::1000::gconfd-2 3230 195
j3::1000::gdbus 13171 194
j3::1000::gdbus 18507 194
j3::1000::gdbus 2358 195
j3::1000::gdbus 2634 195
j3::1000::gdbus 2657 195
j3::1000::gdbus 2667 195
j3::1000::gdbus 2686 195
j3::1000::gdbus 2698 195
j3::1000::gmain 6802 195
j3::1000::gmain 8332 195
j3::1000::gnome-keyring-d 2358 195
j3::1000::gnome-pty-helpe 6812 195
j3::1000::gnome-session-b 2815 195
j3::1000::gnome-software 3060 195
j3::1000::gpg-agent 2796 195
j3::1000::gvfs-afc-volume 3157 390
j3::1000::gvfsd 2657 195
j3::1000::gvfsd-dnssd 3399 195
j3::1000::gvfsd-fuse 2667 780
j3::1000::gvfsd-metadata 3204 195
j3::1000::gvfsd-network 3354 195
j3::1000::gvfsd-smb-brows 3373 195
j3::1000::gvfsd-trash 3190 195
j3::1000::gvfs-fuse-sub 2667 195
j3::1000::gvfs-goa-volume 3132 195
j3::1000::gvfs-gphoto2-vo 3144 195
j3::1000::gvfs-mtp-volume 3139 195
j3::1000::gvfs-udisks2-vo 3095 195
j3::1000::hud-service 2806 195
j3::1000::ibus-daemon 2686 195
j3::1000::ibus-dconf 2698 195
j3::1000::ibus-engine-sim 2781 195
j3::1000::ibus-ui-gtk3 2699 195
j3::1000::ibus-x11 2701 195
j3::1000::indicator-appli 2931 195
j3::1000::indicator-bluet 2890 195
j3::1000::indicator-datet 2894 390
j3::1000::indicator-keybo 2896 195
j3::1000::indicator-messa 2889 195
j3::1000::indicator-power 2891 195
j3::1000::indicator-print 2901 195
j3::1000::indicator-sessi 2903 195
j3::1000::indicator-sound 2899 195
j3::1000::io_compute 8332 194
j3::1000::io_worker 8332 388
j3::1000::nautilus 3067 195
j3::1000::nm-applet 3047 195
j3::1000::notify-osd 13171 194
j3::1000::plugin_host 8349 194
j3::1000::polkit-gnome-au 3057 195
j3::1000::pool 2634 17
j3::1000::pool 4476 1560
j3::1000::process_status 8332 194
j3::1000::process_status 8349 194
j3::1000::pulseaudio 2966 195
j3::1000::sd_cicero 9586 388
j3::1000::sd_dummy 9583 388
j3::1000::sd_espeak 9578 776
j3::1000::sd_generic 9590 388
j3::1000::(sd-pam 2352 195
j3::1000::sh 3304 195
j3::1000::shm_reader 8332 194
j3::1000::speech-dispatch 9593 388
j3::1000::sublime_text 8332 195
j3::1000::syndaemon 2971 195
j3::1000::systemd 2351 195
j3::1000::threaded-ml 9578 194
j3::1000::threaded-ml 9583 194
j3::1000::threaded-ml 9586 194
j3::1000::threaded-ml 9590 194
j3::1000::thread_queue 8349 194
j3::1000::timer 2358 195
j3::1000::unity-fallback- 3049 195
j3::1000::unity-files-dae 4476 195
j3::1000::unity-panel-ser 2830 195
j3::1000::unity-scope-hom 4463 195
```

```
j3::1000::unity-scope-loa 4474 195
j3::1000::unity-settings- 2808 195
j3::1000::unity-video-len 18507 194
j3::1000::update-notifier 3511 195
j3::1000::upstart 2360 195
j3::1000::upstart-dbus-br 2615 195
j3::1000::upstart-dbus-br 2632 195
j3::1000::upstart-file-br 2627 195
j3::1000::upstart-udev-br 2525 195
j3::1000::/usr/bin/x-term 6802 195
j3::1000::window-stack-br 2575 195
j3::1000::zeitgeist-daemo 3308 195
j3::1000::zeitgeist-datah 3297 195
j3::1000::zeitgeist-fts 3315 195
j3::1000::zsh 10180 193
j3::1000::zsh 6813 195
messagebus::106::dbus-daemon 1084 195
nobody::65534::dnsmasq 2441 195
openldap::121::slapd 1843 780
root::0::accounts-daemon 1013 195
root::0::acpid 1157 195
root::0::acpi_thermal_pm 128 195
root::0::agetty 1814 195
root::0::amdgpu_cs:0 1286 195
root::0::aspell 10670 5
root::0::aspell 10688 4
root::0::aspell 10705 1
root::0::ata_sff 74 195
root::0::bash 10215 191
root::0::bash 10233 171
root::0::bash 10243 168
root::0::bash 10253 164
root::0::bash 10365 81
root::0::bash 10414 71
root::0::bash 10428 66
root::0::bash 10445 63
root::0::bash 10617 42
root::0::bash 10628 40
root::0::bluetoothd 1147 195
root::0::cfg80211 713 195
root::0::charger_manager 160 195
root::0::cleanup 3105 195
root::0::comp_1.0.0 234 195
root::0::comp_1.0.1 238 195
root::0::comp_1.1.0 235 195
root::0::comp_1.1.1 239 195
root::0::comp_1.2.0 236 195
root::0::comp_1.2.1 240 195
root::0::comp_1.3.0 237 195
root::0::comp_1.3.1 241 195
root::0::cpuhp/0 12 195
root::0::cpuhp/1 13 195
root::0::cpuhp/2 19 195
root::0::cpuhp/3 25 195
root::0::cpuhp/4 31 195
root::0::cpuhp/5 37 195
root::0::cpuhp/6 43 195
root::0::cpuhp/7 49 195
root::0::cron 1014 195
root::0::crypto 69 195
root::0::cups-browsed 1225 195
root::0::cupsd 1042 195
root::0::dbus-daemon 782 195
root::0::dbus-launch 781 195
root::0::dconf-service 797 195
root::0::devfreq_wq 77 195
root::0::dhclient 6060 195
root::0::disk_cache:0 1286 195
root::0::dm_timer_queue 249 195
root::0::docker-containe 1879 3120
root::0::docker-containe 4160 1950
root::0::dockerd 1757 3315
root::0::docker-proxy 4155 2340
root::0::ecryptfs-kthrea 85 195
root::0::edac-poller 76 195
root::0::ext4-rsv-conver 291 195
root::0::fwupd 3125 390
root::0::gallium_drv:0 1286 195
root::0::gconfd-2 5716 195
```

```
root::0::gdbus 1013 195
root::0::gdbus 3105 195
root::0::gdbus 3125 195
root::0::gdbus 797 195
root::0::gfx 233 195
root::0::git 10553 22
root::0::git 10555 20
root::0::git 10566 18
root::0::git 10660 5
root::0::git 10678 4
root::0::git 10695 1
root::0::gmain 1013 195
root::0::gmain 10209 191
root::0::gmain 1027 195
root::0::gmain 1135 195
root::0::gmain 797 195
root::0::gnome-pty-helpe 10214 191
root::0::GUsbEventThread 3125 195
root::0::i915/signal:0 227 195
root::0::i915/signal:1 228 195
root::0::i915/signal:2 229 195
root::0::i915/signal:4 230 195
root::0::InputThread 1286 195
root::0::iprt-VBoxTscThr 1273 195
root::0::iprt-VBoxWQueue 1269 195
root::0::ipv6_addrconf 133 195
root::0::irq/140-mei_me 644 195
root::0::irq/141-iwlwifi 720 195
root::0::irq/19-mmc0 222 195
root::0::irqbalance 1213 195
root::0::jbd2/nvme0n1p1- 290 195
root::0::kauditd 58 195
root::0::kblockd 71 195
root::0::kcompactd0 66 195
root::0::kdevtmpfs 55 195
root::0::khugepaged 68 195
root::0::khungtaskd 63 195
root::0::kintegrityd 70 195
root::0::krfcommd 1650 195
root::0::ksmd 67 195
root::0::ksoftirqd/0 7 195
root::0::ksoftirqd/1 16 195
root::0::ksoftirqd/2 22 195
root::0::ksoftirqd/3 28 195
root::0::ksoftirqd/4 34 195
root::0::ksoftirqd/5 40 195
root::0::ksoftirqd/6 46 195
root::0::ksoftirqd/7 52 195
root::0::kstrp 142 195
root::0::kswapd0 81 195
root::0::kthreadd 2 195
root::0::kthrotld 127 195
root::0::ktpacpid 707 195
root::0::kworker/u16:2 9511 117
root::0::kworker/u16:3 10158 194
root::0::kworker/u16:4 8329 195
root::0::kworker/u17:0 82 195
root::0::kworker/u17:1 787 195
root::0::less 10556 20
root::0::less 10567 18
root::0::lightdm 1244 195
root::0::lightdm 2338 195
root::0::loop0 369 195
root::0::loop1 371 195
root::0::loop2 372 195
root::0::loop4 387 195
root::0::loop5 389 195
root::0::loop6 20913 194
root::0::lsof 10178 194
root::0::lsof 10179 1
root::0::md 75 195
root::0::migration/0 10 195
root::0::migration/1 15 195
root::0::migration/2 21 195
root::0::migration/3 27 195
root::0::migration/4 33 195
root::0::migration/5 39 195
root::0::migration/6 45 195
root::0::migration/7 51 195
```

```
root::0::mm_percpu_wq 6 195
root::0::ModemManager 1027 195
root::0::mongo 10314 38
root::0::mongo 10319 54
root::0::mongo 10494 38
root::0::mongo 10532 22
root::0::mongo 10537 16
root::0::netns 56 195
root::0::NetworkManager 1135 195
root::0::nmap 10294 2
root::0::nmap 10303 7
root::0::nmap 10304 7
root::0::nmap 10305 7
root::0::nmap 10306 12
root::0::nmap 10581 6
root::0::nmap 10746 2
root::0::nmbd 1924 195
root::0::nvme-wq 218 195
root::0::oom_reaper 64 195
root::0::pager 10554 22
root::0::ping 300 195
root::0::polkitd 1227 195
root::0::pool 1135 133
root::0::pool 3105 6
root::0::probing-thread 3105 195
root::0::python3 10177 194
root::0::rcu_bh 9 195
root::0::rcu_sched 8 195
root::0::rcu_tasks_kthre 57 195
root::0::scp 10438 30
root::0::scsi_eh_0 216 195
root::0::scsi_eh_1 219 195
root::0::scsi_tmf_0 217 195
root::0::scsi_tmf_1 220 195
root::0::sdma0 242 195
root::0::sdma1 243 195
root::0::sh 10375 81
root::0::sh 10424 69
root::0::sh 10427 68
root::0::sh 10440 64
root::0::sh 10616 43
root::0::SignalSender 10314 19
root::0::SignalSender 10319 27
root::0::SignalSender 10494 19
root::0::SignalSender 10532 11
root::0::SignalSender 10537 8
root::0::si_shader:0 1286 195
root::0::si_shader:1 1286 195
root::0::si_shader:2 1286 195
root::0::si_shader_low:0 1286 195
root::0::si_shader_low:1 1286 195
root::0::smbd 10583 4
root::0::smbd 10584 2
root::0::smbd 10748 1
root::0::smbd 10749 1
root::0::smbd 1967 195
root::0::smbd 1968 195
root::0::smbd 1975 195
root::0::snapd 1020 5070
root::0::sqlmap 10358 74
root::0::sqlmap 10407 60
root::0::ssh 10439 30
root::0::sshd 1247 195
root::0::su 10627 40
root::0::sudo 10176 194
root::0::sudo 10208 192
root::0::systemd 1 195
root::0::systemd-journal 320 195
root::0::systemd-logind 1025 195
root::0::systemd-udevd 365 195
root::0::thermald 1024 390
root::0::tracepath 10263 10
root::0::tracepath 10264 11
root::0::tracepath 10267 13
root::0::tracepath 10268 5
root::0::tracepath 10269 27
root::0::tracepath 10272 8
root::0::tracepath 10273 27
root::0::ttm_swap 232 195
```

```
root::0::udisksd 3105 195
root::0::unattended-upgr 1222 195
root::0::upowerd 1656 195
root::0::/usr/bin/termin 10209 191
root::0::uvd 244 195
root::0::uvd_enc0 245 195
root::0::uvd_enc1 246 195
root::0::vce0 247 195
root::0::vce1 248 195
root::0::vi 10638 24
root::0::watchdog/0 11 195
root::0::watchdog/1 14 195
root::0::watchdog/2 20 195
root::0::watchdog/3 26 195
root::0::watchdog/4 32 195
root::0::watchdog/5 38 195
root::0::watchdog/6 44 195
root::0::watchdog/7 50 195
root::0::watchdogd 78 195
root::0::winbindd 1925 195
root::0::winbindd 1937 195
root::0::winbindd 1973 195
root::0::winbindd 1974 195
root::0::wish 10661 10
root::0::wish 10679 8
root::0::wish 10696 2
root::0::wpa_supplicant 1347 195
root::0::writeback 65 195
root::0::Xorg 1286 195
root::0::zsh 10328 107
root::0::zsh 10376 79
root::0::zsh 10455 61
root::0::zsh 10752 3
rtkit::118::rtkit-daemon 1626 585
syslog::104::in:imklog 1136 195
syslog::104::in:imuxsock 1136 195
syslog::104::rs:main_Q:Reg 1136 195
syslog::104::rsyslogd 1136 195
systemd-timesync::100::sd-resolve 534 195
systemd-timesync::100::systemd-timesyn 534 195
uuidd::107::uuidd 13255 194
whoopsie::109::gdbus 1760 195
whoopsie::109::gmain 1760 195
whoopsie::109::whoopsie 1760 195
```